Departamento de Informática

PROYECTO FIN DE CARRERA

CONSIDERACIONES A MODO DE GUÍA A PARTIR DE ESTÁNDARES IEEE PARA CALIDAD Y BUENAS PRÁCTICAS EN DESARROLLO DE SOFTWARE.

Autor: María del Pilar Martínez de Morentin Góngora

Tutor: Miguel Ángel Ramos González

Leganés, octubre de 2011

Título: Consideraciones a modo de guía a partir de estándares IEEE para calidad y buenas prácticas en desarrollo de software

Autor: María del Pilar Martínez de Morentin Góngora

Director: Miguel Ángel Ramos González

EL TRIBUNAL

Presidente: Luis García Sánchez

Vocal: Pilar *Aranzazu Herráez*

Secretario: Antonio García Carmona

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 27 de Octubre de 2011en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimiento

He de expresar mi profundo agradecimiento a aquellas personas que me han ayudado a realizarme tanto intelectual como personalmente ya que su ayuda ha sido de gran importancia para poder llegar al sitio que me encuentro.

Este proyecto es el cierre de una puerta para abrirse otra, sin dejar de mencionar a aquellas personas que lo han hecho posible, ya que el camino ha sido muy largo pero a la vez lleno de satisfacciones. Y por ende quiero dejar constancia de mi agradecimiento.

En un lugar muy destacado quiero agradecer a mis padres ya que han sido el eje principal para la culminación de esta etapa, sin su apoyo esto no sería posible, a mi padre por enseñarme a valorar que en lo pequeño está lo grande, a mi hermano Juan por su apoyo incondicional, ánimo, ayuda; a mi hermana y Enrique por su comprensión y a mi sobrinito Aarón por darme tantos momentos de alegría.

Un agradecimiento a mi familia, en especial a mi abuela y tíos: Esperanza, María Ángeles, Valentina, Asunción… ya que siempre estuvieron allí en los momentos que más les necesité.

A mi tutor Miguel Ángel por haberme dado la oportunidad de realizar este proyecto con él. Su calidad humana, entrega, servicio, ánimo, por su infinita paciencia y me quedaría corta al hablar de él, cuya importancia ha sido vital en la realización de este proyecto.

Un agradecimiento en especial a mis compañeros Iván Rodríguez y Diego Carballés por su valor incalculable y por haber tenido la suerte de conocerles, a Juan María Egido muchas gracias por todos los favores que he recibido y demás compañeros como Dalila, Chumo, Laura, Pedro…

Quiero agradecer a todos los profesores que me han aportado un granito de arena en mi formación y en especial hacer mención a los profesores Luis García, Manuel Velasco por su ayuda y comprensión.

Por supuesto a Domingo Gaitero por haberme dado la oportunidad de estar en una de sus conferencias y facilitarme documentación para la realización de este proyecto.

Mil gracias a todos.

Resumen

Mediante este proyecto a través de la selección de estándares (IEEE) tomados los más vigentes en las distintas fases de desarrollo del software como son: plan de aseguramiento de la calidad del software, especificación de requisitos, descripción de diseño del software, fundamentos de pruebas, se han unificado en una guía para que empresas u organizaciones puedan elevar el nivel de calidad, disminuir costes y aumentar la productividad, formando parte de la política y de los objetivos estratégicos de toda organización o empresa que tenga visión de futuro, dado que con ello se satisface al cliente, se generan beneficios.

Esta guía proporciona a empresas u organizaciones que desean desarrollar productos o servicios de calidad una estructura adecuada, desde que el cliente piensa en lo que desea es calidad, hasta la entrega del producto o servicio es calidad.

Por ello, su previsión fundamental debe ser la de proveer productos o servicios o realizar actividades, que satisfagan las necesidades expresadas e implícitas de los clientes todo ello al menor coste posible.

El despliegue de la función de la calidad es un proceso estructurado y riguroso capaz de identificar y transmitir la voz del cliente para transformarla en requisitos del producto o servicio, a lo largo de las diferentes etapas que constituyen el desarrollo de dicho producto o servicio, contando con la contribución de todo el personal implicado.

Resulta un modo adecuado en el control cotidiano del proceso de diseño de productos y servicios, por el enfoque sistemático y organizado para que los deseos del cliente sean tenidos en cuenta a la hora de diseñar nuevos productos o servicios.

La transmisión de las necesidades del cliente se realiza a través de todo el proceso de desarrollo del producto, mediante el despliegue sistemático de las relaciones entre las necesidades y características. Estas exigencias se convierten, paso a paso, en propiedades y metas importantes de diseño y en apoyos para asegurar la calidad en la fase de producción.

La funcionalidad de la calidad se basa en la orientación hacia el cliente, con una gestión de la calidad moderna, en todas las fases de desarrollo de un producto o servicio y todo ello si se sigue adecuadamente la guía ya que las expectativas o deseos del cliente o usuario dirigen todo el proceso creativo y son los parámetros claves del proyecto de realización del ingeniero, que actúa como mediador entre las exigencias del cliente y las soluciones viables.

Palabras clave: Ingeniería del software, IEEE, calidad, plan de aseguramiento de la calidad del software, especificación de requisitos, diseño, pruebas.

Abstract

Through this project and through the selection of standards (IEEE) taken the most force in the different phases of software development including: assurance plan software quality, requirements specification, software design description, test basics have been unified into a guide for companies or organizations can raise quality, lower costs and increase productivity, is part of the political and strategic objectives of any organization or company that has vision, because with it satisfies the customer, are profitable.

This guide provides businesses or organizations wishing to develop quality products or services an appropriate structure from which the client think about what you want is quality, to delivery of product or service quality.

Therefore, its fundamental provision should be to provide products or services or activities that meet the needs expressed and implied customer all at the lowest possible cost.

The deployment of quality function is a structured and rigorous process able to identify and communicate the voice of the customer to transform it into a product or service requirements, throughout the different stages that make up the development of that product or service, counting with the contribution of all staff involved.

It is an appropriate way in the daily control of the design process of products and services, systematic and organized approach to the customer's wishes are taken into account when designing new products or services.

The transmission of customer needs is through the entire product development process through the systematic deployment of the relationship between the needs and characteristics. These requirements become, step by step, important goals of properties and design and support for quality assurance in the production phase.

The functionality of quality is based on customer orientation, with a modern quality management in all phases of development of a product or service and all that if you follow the guide properly and that customer expectations or desires target user or the entire creative process and are the key parameters of the project to develop the engineer, who acts as mediator between customer requirements and feasible solutions.

Keywords: Software Engineering, IEEE, quality assurance plan software quality, requirements specification, design, testing.

Índice general

Índice de figuras

Calidad en el Software

Modelos

Plan de Aseguramiento de la Calidad del Software

Especificación de Requisitos del Software

Descripción de Diseño del Software

Fundamentos de Prueba del Software

Índice de tablas

Modelos

Plan de Aseguramiento de la calidad del Software

Especificación de Requisitos del Software

Descripción del Diseño del Software

Fundamentos de Pruebas del Software

Capítulo 1

1 Introducción y objetivos

1.1 Introducción

Se han utilizado en el proyecto estándares (IEEE) en el cual se han unificado en una guía para que empresas u organizaciones que deseen ofrecer calidad, se encuentren en la necesidad de establecer estándares que especifiquen de un modo claro y preciso, la metodología con que se han de desarrollar las diferentes actividades implicadas durante las fases del ciclo del producto o servicio.

Las actividades implicadas en todas las fases del ciclo del producto se implantan mediante un proceso a través del cual se unifican criterios respecto a determinadas áreas y se posibilita la utilización de un lenguaje común en un determinado campo de actuación, esto conlleva a establecer y llegar a acuerdos relacionados con características técnicas que han de cumplir los productos o servicios.

Mediante esta guía se pretende establecer de una manera clara y precisa cada una de las fases desarrolladas enfocadas a la calidad, porque, si se desea producir buena calidad para el consumidor, es necesario decidir por adelantado qué calidad de producto o servicio se ha de planificar (calidad de diseño), producir (calidad de fabricación) y vender calidad que desea el cliente.

La calidad, es cada vez más demandada por el mercado, de tal modo, que sólo las empresas que produzcan con calidad tienen futuro. Los estándares adquieren un gran protagonismo, dada la íntima relación que tiene la calidad con la estandarización, puesto que esta, es el patrón y el fundamento de una calidad que alude al desarrollo del producto o servicio y su grado de cumplimiento respecto a las especificaciones previamente establecidas.

Vivimos en un mundo cambiante, en que los modelos del pasado ya no garantizan el futuro. Se ha pasado de la tierra, el capital y el trabajo, como agentes de competitividad, a los conceptos de conocimiento e innovación.

La situación actual del mercado exige a las empresas u organizaciones establecer políticas cuyo objetivo prioritario sea mejorar la competitividad. Para ello sus productos o servicios han de responder a los requisitos del cliente al

mejor precio posible, con el fin de mantener y mejorar los beneficios y la posición en el mercado.

Las empresas u organizaciones necesitan unas herramientas o metodologías que les ayuden a resolver el problema planteado, dando lugar a la aparición de las metodologías de ingeniería de la calidad. Por medio de estas metodologías de ingeniería de la calidad, las empresas u organizaciones pueden seguir avanzando en la mejora continua, dado que posibilitan y facilitan la previsión y resolución de problemas planteados, sobre todo en las fases iniciales de diseño.

Nos enfrentamos a una situación con dos perspectivas diferentes. Por un lado las empresas u organizaciones desean desarrollar y entregar software confiable, a tiempo y de acuerdo al presupuesto del cliente. Por otro lado está el cliente a la expectativa de que todo lo anterior se cumpla. Por eso las organizaciones u empresas deben buscar una lista de verificación de cosas que hacer y la búsqueda es en particular útil para asegurar que están cubiertas todas las bases de la calidad: de ahí la utilidad de esta guía, que puede ayudarles a conseguir su meta de la calidad, es decir, la competitividad.

La competitividad no es la única razón que se busca en la calidad del software. Por ende debemos dar importancia al desarrollo del producto o servicio, debemos ser conscientes y responsables de las consecuencias que producirían los defectos que el producto o servicio podrían ocasionar. Las pérdidas relativas a la calidad son las pérdidas ocasionadas por la no aplicación del potencial de recursos en procesos y en actividades. Como pérdidas relativas a la calidad pueden incluirse las relacionadas con la insatisfacción del cliente, el despilfarro de recursos y materiales, la pérdida de poder aumentar el valor añadido para los clientes, vidas humanas, etc.

¿Por qué es importante esta guía? Es importante porque da confianza, es decir, permite obtener un nivel de ordenamiento óptimo; fiabilidad; seguridad de funcionamiento; garantía de calidad de los elementos que se están desarrollando y sobre todo ayuda a trabajar en cada momento con responsabilidad porque están en nuestras manos muchos factores tanto internos como externos.

Además esta guía es importante para que empresas u organizaciones tengan un enfoque de la integridad y desarrollo del producto o servicio siempre con miras a la satisfacción del cliente y el redescubrimiento de la importancia de la calidad del producto o servicio en sus distintas vertientes.

1.2 Objetivos

El aumento de la competencia junto con una clientela cada vez mejor informada ha convertido la satisfacción del cliente en una de las principales preocupaciones de empresas u organizaciones.

El objetivo de este proyecto es conseguir que las empresas u organizaciones que se dedican al desarrollo del software tengan a su alcance una guía que reúna todas las fases del desarrollo software. Esta guía está enfocada al desarrollo y evolución de un producto o servicio desde la concepción hasta la entrega del producto mediante métodos, herramientas y técnicas para facilitar un trabajo competente y de alta calidad.

A través de esta guía se pretende alcanzar los siguientes objetivos:

- Con esta guía se alcanza a comprender:

    - Simplificación: Minimizar los modelos para quedarse estrictamente con los necesarios.
    - Unificación: Permite el intercambio a nivel internacional.
    - Especificación: Se busca evitar errores de identificación creando un modo de lenguaje claro y preciso.

- Aumentar la relación y el intercambio de ideas. Del tal modo que todas las partes implicadas en el desarrollo del producto o servicio deben ser tenidas en cuenta por los estándares que las definen, para lo que es preciso comunicación, colaboración eficaz entre ellas.

- Suprimir barreras comerciales, a partir de unos mínimos de calidad y de seguridad en los productos o servicios. El hecho de que existan estándares que se conozcan o apliquen, facilita tanto el comercio interno como externo, entre otras razones porque existen un lenguaje común y una adaptación de los productos a estándares internacionales.

- Resaltar y explicar la importancia de la calidad en cada fase de desarrollo software.

  Se proponen los siguientes objetivos parciales:

    - Comprobar la eficacia del sistema de la calidad implantado para alcanzar los objetivos de la calidad específicos.
    - Especificaciones, uniformidad, eficiencia, estética.
    - Resultados esperados y seguridad en el funcionamiento.
    - Las diversas etapas en los procesos. Con asignación de responsabilidades y recursos.
    - Los procedimientos e instrucciones documentados, además de los programas de prueba, inspección y auditoría donde fuese necesario.

- Conseguir la satisfacción de los clientes.

  Se proponen los siguientes objetivos parciales:

    - Conocer las necesidades y deseos de los clientes, para satisfacerlos inmediatamente.

- Medir y analizar la satisfacción de los clientes. Para que el cliente se sienta más satisfecho se debe mejorar el producto o servicio realizado o bien generar expectativas más realistas.

Las soluciones sencillas son siempre las mejores, por ello con un lenguaje simple debemos ser capaces de explicar qué es un estándar, en qué consiste la calidad y utilización de los distintos modelos, herramientas y técnicas empleados en los estándares y la calidad.

1.3 Fases de desarrollo

Para llevar a cabo el proyecto he tenido que realizar los siguientes pasos:

Buscar la documentación pertinente relacionada con los estándares de IEEE, ya que al unificar en una sola guía las fases de desarrollo del software, he seleccionado los estándares más vigentes.

En cada uno de los estándares como son: plan de aseguramiento de la calidad del software, especificación de requisitos software, descripción del diseño de software, fundamento de las pruebas del software; en cada uno de ellos he realizado los siguientes pasos:

- Traducir cada uno de los estándares mencionados anteriormente al castellano.
- Leer cada uno de los estándares.
- Buscar información para desarrollar cada uno de ellos mediante: libros en castellano, libros en inglés, referencias con otros estándares (IEEE). Con todos estos medios he recopilado información y la he plasmado en el desarrollo de cada uno de los estándares.

1.4 Objetivos principales

El objetivo fundamental de este proyecto es conseguir un producto o servicio de alta calidad mediante una guía que reúne las fases del ciclo de desarrollo del software para una mayor integridad y comodidad de organizaciones o empresas. En base a estos objetivos principales:

- Introducir en la gestión de la calidad y medición del software.

  Se proponen los siguientes objetivos parciales:

  - Comprender el proceso de gestión de la calidad y las actividades del proceso, planificación y control de la calidad.
  - Comprender la importancia de los estándares en el proceso de gestión de calidad.

- Comprender qué son las métricas de la calidad.
- Comprender como las mediciones pueden ser necesarias en el cálculo de los atributos de calidad de software.

- Describir la labor de análisis y definición de los requisitos que ha de cumplir un proyecto de software.

  Se proponen los siguientes objetivos parciales:

  - Describir los fines y objetivos del software a desarrollar.
  - Comprender por qué las técnicas de especificación formal ayudan a descubrir problemas en los requisitos del sistema.
  - Comprender por qué las técnicas formales basadas en modelos formales se usan para especificar el comportamiento.
  - Obtener las especificaciones que debe cumplir el sistema a desarrollar. El medio que se emplea para lograr dicho objetivo es obtener un modelo válido, necesario y suficiente para recoger todas las necesidades y exigencias que el cliente precisa del sistema y restricciones que el analista considere que debe verificar el sistema.

- Introducir varios modelos de sistemas que puedan desarrollarse durante el proceso de los requisitos.

  Se proponen los siguientes objetivos parciales:

  - Comprender por qué es importante establecer los límites de un sistema y modelar su contexto.
  - Comprender los conceptos del modelado de comportamiento, modelado de los datos y modelado de los objetos.
  - Introducir en algunas de las notaciones definidas en el lenguaje unificado de modelado (UML) y como estas notaciones pueden usarse para desarrollar modelos de sistemas.

- Analizar el diseño arquitectónico y diseño detallado

  Se proponen los siguientes objetivos parciales:

  - Comprender por qué es importante el diseño arquitectónico y diseño detallado del software.
  - Comprender las decisiones que deben tomarse sobre la arquitectura del sistema durante el proceso de diseño arquitectónico.
  - Comprender las distintas técnicas del diseño de software que proporciona la organización del sistema en su totalidad, descomposición modular, técnicas basadas en abstracciones, técnicas de diseño de datos, etc.

- Describir el proceso de pruebas del software e introducir varias técnicas de pruebas.

Se proponen los siguientes objetivos parciales:

- Comprender los principios de las pruebas del sistema y las pruebas de componentes.
- Comprender las estrategias que pueden utilizarse para generar casos de pruebas del sistema.
- Descubrir los errores y fallos cometidos durante las fases anteriores de desarrollo del producto.
- Analizar las técnicas de verificación y validación

  Se proponen los siguientes objetivos parciales:

  - Entender cómo puede medirse la fiabilidad del software y cuando será alcanzado un nivel requerido de fiabilidad.
  - Comprender los principios de los argumentos de seguridad y como pueden utilizarse.

1.5 Estructura de la memoria

Para facilitar la lectura de la memoria, se incluye a continuación un breve resumen de cada capítulo.

En el capítulo 2, se incluyen definiciones y argumentos para comprender la importancia de la calidad de la empresa u organización, sus problemas y factores constituyentes, su evolución y el futuro de la calidad.

En el capítulo 3, se proporciona un marco general de referencia según el modelo de madurez de la capacidad del software, con el fin de establecer un programa de medición que se pueda usar para medir sistemáticamente el progreso de los productos y de los procesos de desarrollo del software.

En el capítulo 4, se habla de la unificación de los estándares, el cual está compuesto de:

- **Aseguramiento de la calidad del software**

  Este estándar, gran parte de él está dirigido a proyecto grande, pero también se puede usar en los pequeños y el estándar ayuda a recordar todos los factores que deben incluirse. Entre ellos cabe mencionar: Quién será responsable de la calidad, que documentación se requiere, que técnica se usará para asegurar la calidad, qué procedimientos se seguirán para administrar el proyecto, pruebas, informe de problemas y acción correctiva, herramientas técnicas y metodologías, control de proveedores, capacitación, gestión de riesgos. Se dirige hacia el desarrollo y mantenimiento de software.

- **Especificación de requisitos**

  Se describe la labor de análisis y definición de los requisitos que ha de cumplir un proyecto de software. Esta labor debe dar lugar a la especificación de software, en la que se concretan las necesidades que se desean cubrir y se fijan las restricciones con las que debe trabajar el software a desarrollar. El análisis se realiza dentro de la primera fase (fase de definición) del ciclo de vida y tiene una importancia decisiva para conseguir que el sistema que se construya finalmente sea realmente el que se deseaba.

- **Descripción de diseño del software**

  En el diseño se inicia en el estudio de las etapas de desarrollo. Después de haber especificado QUÉ se quiere resolver durante la especificación, las etapas de desarrollo se dedican a determinar CÓMO se debe resolver el proyecto. La primera de estas etapas es la del diseño, se continúa con la de codificación y se finaliza con las etapas de pruebas del software realizado.

  Primeramente se concreta qué se entiende por diseño y se analizan las actividades que se deben realizar para llevarlo a cabo. A continuación se introducen algunos conceptos fundamentales que deben tenerse en cuenta para realizar cualquier diseño. Algunas de estas notaciones son versiones ampliadas de las ya estudiadas para la especificación y otras son totalmente especificadas para el diseño.

- **Fundamentos de las pruebas de software**

  El propósito del software basado en pruebas de sistemas es ayudar a la organización de desarrollo de la calidad a la construcción del software y al sistema durante los procesos del ciclo de vida y para comprobar que la calidad se ha logrado.

  El proceso de prueba determina si los productos de una actividad del ciclo de vida determinado se ajustan a los requisitos de esa actividad, y si el producto cumple el uso previsto y las necesidades del usuario. Esta determinación puede incluir la inspección, demostración, análisis y prueba de productos de sistemas de software basados en software.

En el capítulo 5, el presupuesto, incluye división de las distintas fases en que está compuesto el proyecto; mediante el Project se observará en el diagrama de Gantt; desglose de coste de personal; coste material y coste total.

En el capítulo 6, glosario, incluye aquellos términos que son pocos conocidos o de difícil interpretación.

En el capítulo 7, bibliografía, fuentes documentales, estándares, bibliografías utilizadas en el texto.

En el capítulo 8, se resumen las principales conclusiones del trabajo, indicando el cómo y el porqué el proyecto puede avanzar en fases futuras.

Capítulo 2

2 Calidad en el software

2.1 Introducción: Definición y alcance de la calidad

“Olvidar el precio de un producto es reconocer su calidad“ “Frederic Henry Royce, confundador de Rolls Royce.

La calidad se puede definir como “el conjunto de las propiedades de un producto, procesos o servicio que le confieren aptitud para satisfacer las necesidades declaradas o implícitas del cliente”.

La gestión de la calidad organiza la planificación, la ejecución y el control de la calidad, así como su dirección, demostración, auditoría y mejora continua. La ingeniería de calidad se concentra en obtener productos y procesos de calidad aplicando equipos y métodos con cualidades tecnológicas y económicas óptimas.

El comportamiento social en el competitivo mercado actual, muy orientado hacia el consumidor, revela que la estrategia de una empresa regida por el lema de Royce promete el éxito, hoy más que nunca. La calidad es cada vez más un factor de competitividad, junto con los factores tradicionales: tiempos y costes.

La calidad, o sus componentes, como la no-depreciación, duración, funcionalidad y prestaciones, etc., determinan la decisión del cliente hacia los bienes de consumo, productos complejos o incluso los servicios. Tiene, por ello, un efecto catalizador sobre las estrategias de mercado mundiales. La calidad acuña la buena fama: quien la posee tiene las puertas abiertas; el que la pierde se hunde rápidamente y sólo con mucha dificultad conseguirá volver a relacionar su nombre con el atributo calidad.

2.2 Desarrollo y situación de la calidad

En los años 50 Japón constata la poca calidad en su línea de fabricación de productos de consumo masivo, y la negativa respuesta del mercado americano y europeo a su estrategia de “cantidad a precios bajos”.

De modo casi simultáneo plantea el problema de la calidad en todos los campos de la actividad industrial. Japón redescubre y aplica así los métodos estadísticos de evaluación y, en especial, el control estadístico de procesos,

desarrollado por expertos americanos en los años 20 y 30. También entonces se asumen los principios de la gestión de la calidad total de los americanos de los años 50.

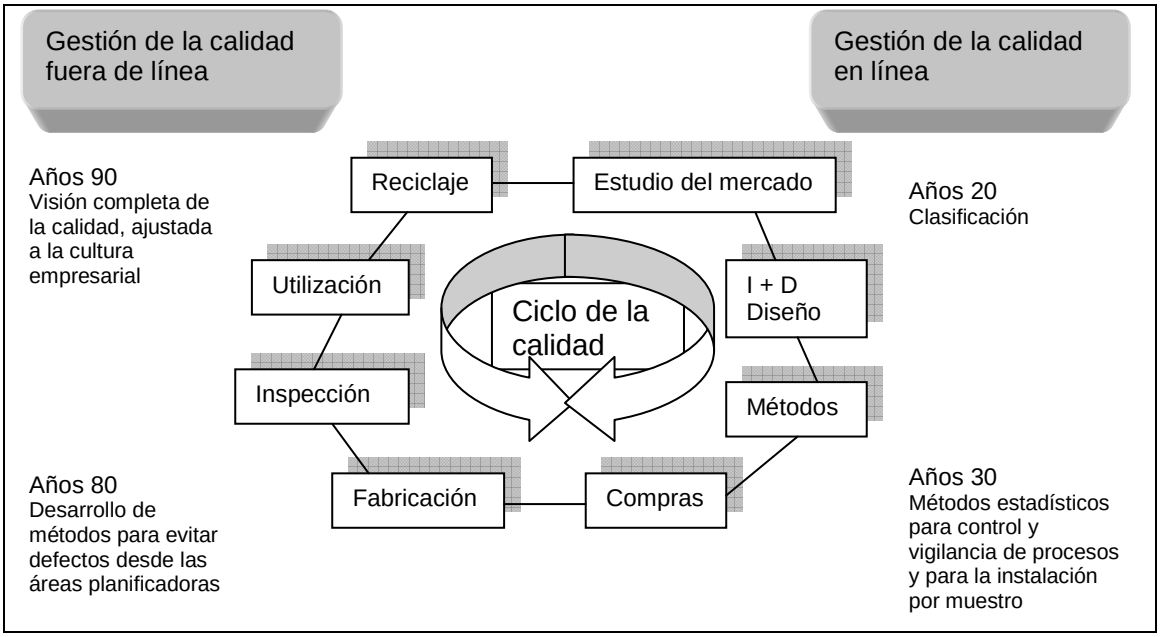


Figura 1.Desde el control del producto a la gestión integrada de la calidad. Adaptado de [Tilo Pfeifer – Fernando Torres]

En los años 70, ya se hablaba del milagro japonés de la calidad, que dominaba claramente muchos mercados con productos de calidad. El análisis de las causas del éxito japonés mostro un resultado sorprendente.

No solo habían conseguido satisfacer de modo óptimo las exigencias y expectativas de los clientes, sino también algo mucho más significativo: los métodos de gestión de la calidad fuera de línea eran extraordinariamente eficaces para ahorrar tiempos y costes. Con ello conseguían fabricar productos de máxima calidad, lo que elevaba el beneficio y permitía ofrecer considerables descuentos a los clientes. Por si fuera poco, el tiempo de desarrollo y puesta en mercado de nuevos productos se reducía. Todos estos factores les aseguraban su competitividad en los mercados mundiales.

El escaso empleo de los métodos hoy disponibles, y su lenta aunque continua e imparable evolución hacia la gestión integrada de la calidad, son también impedidos por las consecuencias de una organización funcional. Esto constata en la filosofía de control de la calidad en línea, sólo dedicado a descubrir fallos, y en el caso y posterior uso de los métodos de gestión de la calidad fuera de línea, con acciones para evitar errores y producir calidad desde la planificación.

A menudo los desarrolladores no se basan tanto en los deseos del mercado, como en lo que el ingeniero considera técnicamente fabricable u óptimo, según los criterios de calidad. El cumplir las exigencias que el cliente impone al producto en lo relativo a su funcionamiento, solidez, estabilidad, diseño, etc., es decir, su idea de la calidad, es más valorado en la compra que el aspecto

económico. Así lo prueban varios estudios sobre la estrategia del mercado, realizados a principios de los 80.

2.3 Evolución del concepto de la calidad

No es exagerado afirmar que la calidad es un factor de supervivencia, especialmente para las pequeñas y medianas empresas. Esta tendencia, iniciada ya hace años, ha experimentado en la segunda mitad de los años 80 una gran aceleración.

La calidad, o mejor dicho, la calidad en un mundo de prestaciones con objetivos cambiantes, son decisivas para el continuo desarrollo de la empresa. El comportamiento del comprador también ha cambiado radicalmente. La simple satisfacción de las necesidades básicas ha sido relegada. La clave de la competencia radica en realizar las exigencias y esperanzas del cliente, lo que define el nivel de calidad del producto.

Las empresas industriales y el amplio sector de los servicios compiten intensamente por la calidad desde hace años. La calidad decide hoy la producción, que sufre un cambio sustancial, desde la orientación hacia el producto hasta la percepción global de la empresa como sistema creador de valor. De ello dependen tanto el crecimiento y el mantenimiento de la empresa, como el de subsectores o de sectores productivos completos.

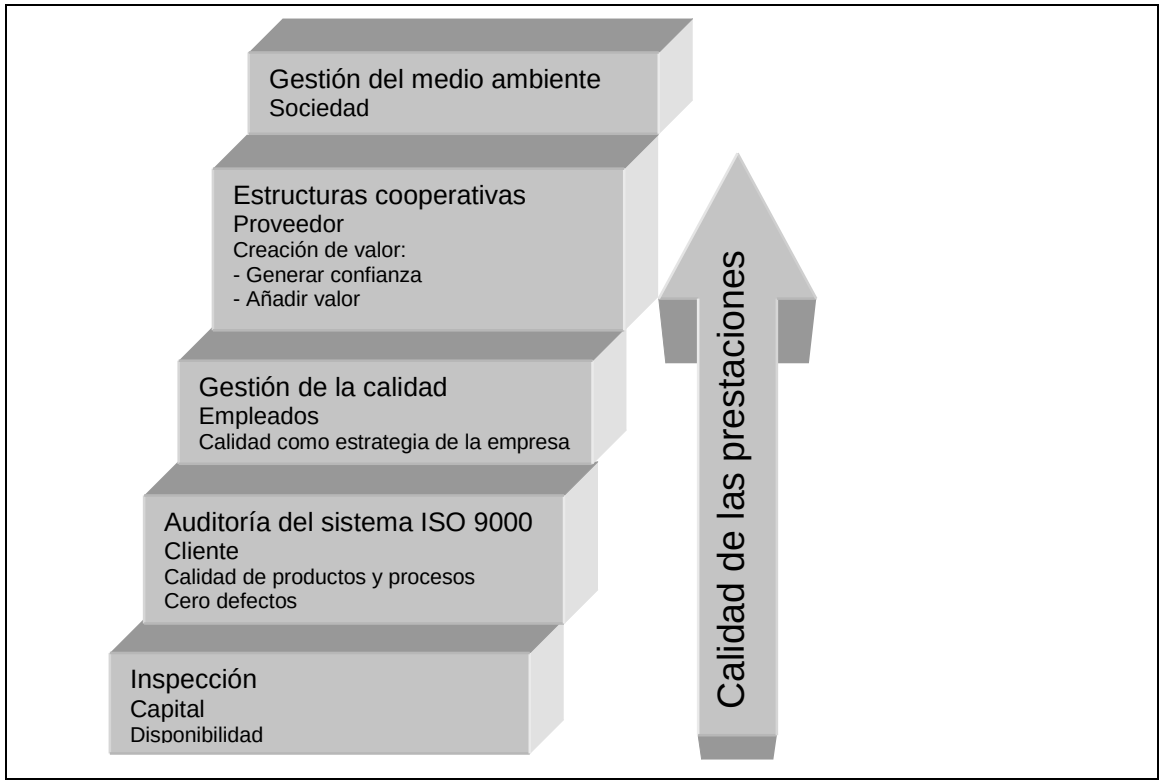


Figura 2.La evolución del concepto de calidad. Adaptado de [Tilo Pfeifer – Fernando Torres]

2.4 Carencias y problemas actuales

A primera vista sorprende que un diagnóstico tan claro de la importancia de la calidad no haga superar con decisión y rapidez el punto débil de la gestión de la calidad hoy practicado. Un análisis rápido y superficial evidencia que apenas hay avance en este sector.

La problemática general de la gestión de la calidad es demasiado compleja: no sólo hay que tener en cuenta diversos principios y áreas de aplicación para las distintas estrategias y métodos de la gestión de la calidad, sino que ello se combina con el empleo de los recursos y medios técnicos, organizativos y personales. También precisan numerosos requisitos para llegar a los objetivos de mejora de calidad y minimizar al mismo tiempo los costes implicados.

- El principal problema de la gestión de calidad es identificar la gestión con el control de la calidad, es decir, que ningún producto defectuoso salga de la fábrica. Se constata a menudo que aproximadamente el 60% de los errores detectados en la industria han aparecido de forma similar alguna otra vez. Ello se explica por el gran distanciamiento temporal y espacial entre el origen y el descubrimiento de los defectos, lo que impide una gestión eficaz de fallos.

- El segundo gran problema de la actual gestión de la calidad resulta así evidente: la ignorancia del significado de los ingresos y costes, o de la disminución de beneficios derivados de la calidad o no calidad. Todavía, y demasiado a menudo, se asocian las acciones de la gestión de calidad con un aumento de coste.

- El tercer gran problema de la industria es el exceso de control. Los compradores tienden cada vez más a condicionar a los proveedores con cifras máximas de defectos residuales del orden de millonésimas. Los proveedores tratan de responder intensificando sus procesos de inspección, para descubrir los productos defectuosos. Por ello, la simple estrategia de control suele fracasar si se exigen fracciones muy pequeñas de fallos, si éstos o sus efectos aún no existen o no se aprecian en el momento de la inspección o si los fallos se deben a causas variables.

- El cuarto problema actual hacia la gestión de calidad, consiste en el escaso conocimiento de los contenidos, eficacia, posibilidades de uso y accesibilidad a los métodos de gestión de la calidad.

Tabla 1.Problemas de la gestión de la calidad

2.5 La calidad en ingeniería del software

Lo primero que se debe considerar a la hora de abordar el tema de la calidad del software es que éste constituye un producto con unas características muy específicas.

- Se desarrolla, no se fabrica en el sentido clásico del término. Todo el coste de su producción se centra en el diseño de la primera copia, ya que la replicación de un programa es una tarea trivial.
- Se trata de un producto lógico, sin existencia física. El verdadero producto del software es el diseño de una serie de instrucciones para el computador. Su existencia en papel (listado) o en soporte magnético (fichero) no es más que una representación en un código o en lenguaje de su autentica naturaleza, las instrucciones. De hecho, está protegido por leyes de propiedad intelectual al igual que la música o las obras escritas.
- No se degrada con el uso. La naturaleza lógica del software permite que permanezca inalterable por muy intensa que sea su utilización. Se puede degradar su representación magnética pero no su esencia.
- La complejidad del software, la ausencia de controles adecuados y el mercado actual lleva a que sea un producto que muchas veces se entrega conscientemente con defectos, incluso públicamente declarados.
- Un porcentaje muy grande de la producción se hace aún a medida, en vez de emplear componentes existentes y ensamblarlos, aunque las bibliotecas de funciones o componentes están ya cambiando en parte esta situación.
- Es muy flexible. Se puede cambiar con facilidad e incluso, se pueden reutilizar trozos de un producto para construir otro. Sin embargo la facilidad para cambiarlo, aunque es un peligro, que hay que controlarlo.

Tabla 2.Adaptación del software en sectores industriales

2.6 Definición de la calidad del software

Según el estándar IEEE std.610-1991 [IEE, 1991a] indica que calidad es el grado con el que un sistema, componente o proceso cumple:

a) Los requisitos especificados.
b) Las necesidades o expectativas del cliente o usuario.

Hay que recordar que:

a) Los requisitos establecidos SRS pueden ser:

    - Requisitos funcionales, que determinan funciones a realizar por el software.
    - Requisitos de otros tipos: De rendimiento, de seguridad, de interfaz, etc.

b) Los estándares y las normas de desarrollo determinan como se debe realizar el proceso de desarrollo de software. Su seguimiento permite que se consiga una calidad técnica en el software producido, que influye en la calidad de cara al usuario, es decir, conformidad con requisitos.

c) Existen requisitos explícitos, no expresamente declarados, es decir, en la SRS o en un contrato; pero que el usuario del software desea obtener. No obstante, es difícil que no existan requisitos implícitos.

2.7 Terminología sobre calidad

Para afrontar el estudio de la calidad del software debemos conocer primero los principales términos empleados en esta área.

Terminología	Descripción
Gestión de la calidad del software	Actividades coordinadas para dirigir y controlar una organización en lo relativo a la calidad, es decir, el aspecto de la función general de la gestión que determina y aplica la política de calidad. La gestión de calidad se aplica a nivel de empresa por lo que incluye planificación estratégica, asignación de recursos etc.
Aseguramiento de la calidad del software	Parte de la gestión de calidad orientada a proporcionar confianza en que se cumplirán los requisitos de la calidad. En el ámbito de software podría presentarse como el conjunto de actividades planificadas y sistemáticas necesarias para aportar la confianza en que el producto software satisfará los requisitos dados de calidad. También puede referirse, en el software, al conjunto de actividades para evaluar el proceso mediante el cual se desarrolla el producto.
Control de la calidad del software	Parte de la gestión de calidad orientada al cumplimiento de los requisitos de la calidad. Incluye las técnicas y actividades de carácter operativo, utilizadas para satisfacer los requisitos relativos a la calidad. Centradas en dos objetivos: mantener bajo control un proceso y eliminar las causas de defectos en las diferentes fases del ciclo de vida.
Verificación y validación del software	Es una actividad ligada al control de la calidad en el ámbito del software: - Verificación: Se trata de comprobar si los productos construidos en una fase del ciclo de vida establecen los requisitos establecidos en la fase anterior. Se suele decir si el producto, hasta el momento, está completo y es consistente. - Validación: Se trata de comprobar si el software construido satisface los requisitos del usuario. Son las actividades de comprobación de que el producto software construido es el que se deseaba construir, es decir, si el producto funciona como el usuario quiere

	y realiza las funciones que se habían solicitado.

Tabla 3.Términos sobre calidad

2.8 Niveles de acción en la calidad del software: Empresa/organización/proyecto

En el ámbito de la calidad del software podemos realizar el siguiente esquema de clasificación de las acciones que se pueden emprender. La actividad en calidad del software puede trabajar en los siguientes ámbitos.

Actividad	Descripción
Calidad de realización.	Se trataría de la mejora de las capacidades individuales de los desarrolladores e ingenieros de software para lograr que realicen mejor las tareas encomendadas, es decir contar con el personal adecuado en cada tarea. La formación apropiada tanto en las tecnologías como en técnicas de desarrollo y mantenimiento y competencias personales tendrá influencia en el rendimiento y calidad obtenida.
Organización de las actividades a nivel de proyecto.	El trabajo en equipo es inherente a la ingeniería de software, no vale un planteamiento únicamente centrado en la mejora de la actividad individual. El siguiente nivel más visible es el proyecto. En este ámbito se contempla tanto la planificación y seguimiento de actividades específicas de calidad.
Organización de las actividades de calidad a nivel de empresa.	Empezamos contemplando la acción a nivel de organización, la mejora de los procesos de desarrollo y mantenimiento de software aportan la madurez necesaria para lograr buenos resultados de eficiencia y calidad.
Análisis por niveles, retos de la calidad programada y la calidad necesaria.	Siempre es necesario resaltar la importancia de un correcto análisis de necesidades del usuario y de una buena especificación de requisitos sin los que es posible aplicar técnica alguna de control de calidad.

Tabla 4.Actividades de la calidad del software

2.9 El futuro de la calidad

Nuevos retos de la gestión de calidad

En los últimos años el gran boom de los sistemas de gestión basados en estándares internacionales no sólo ha afectado al ámbito de la calidad y los sistemas de calidad, sino que ha transcendido a otros ámbitos de la gestión empresarial como el de la gestión medioambiental, la prevención de riesgos

laborales, la responsabilidad social de la empresa o las actividades de innovación en la empresa.

En definitiva, lo que se ha extendido es una forma de trabajar que se ha popularizado con la implantación de los sistemas de calidad, pero que en la actualidad está abriendo nuevos horizontes.

¿Qué es la calidad medioambiental?

La actual preocupación social e institucional por el deterioro medioambiental se ha traducido en presión hacia las empresas para que incorporen un comportamiento más respetuoso con su entorno natural.

El entorno empresarial ha experimentado importantes cambios en este sentido: desde la aparición de un consumidor ecológicamente responsable hasta el desarrollo de una estricta legislación medioambiental, pasando por trabajadores, inversores y vecinos que tienen muy en cuenta el comportamiento social y ecológico de la empresa. Se trata de un entorno que exige a la empresa el diseño de sus objetivos teniendo en cuenta sus dimensiones sociales y ecológicas, que complementen así su dimensión económica.

El término “calidad ambiental” hace referencia a la incorporación, en el ámbito de la gestión empresarial, de una serie de iniciativas que impulsan la minimización de los impactos ambientales negativos de las actividades de las organizaciones.

El término “calidad ambiental” tiene otra razón de ser: en los últimos años una parte de las iniciativas de mejora del impacto ambiental de las empresas ha estado relacionada con la implantación de sistemas de gestión medioambiental basado en estándares internacionales, estándares que se han configurado tomando como base los exitosos estándares internacionales para la implantación de sistemas de gestión de la calidad.

La calidad se ha venido asociando en los últimos años al estándar ISO 9000, la gestión medioambiental se ha asociado a la norma ISO 14000.

¿Qué es la gestión medioambiental? y/o ¿qué es un sistema de gestión medioambiental?

Llamamos gestión medio medioambiental al conjunto de acciones y medidas que se toman en la empresa de cara a contribuir al cumplimiento de la legislación medioambiental vigente y a reducir el impacto medioambiental de la empresa, a través del control de los procesos y actividades que generan dicho impacto medioambiental.

Todas estas acciones y medidas conforman, de forma conjunta, planificada y organizada, el llamado Sistema de Gestión Medioambiental (SGMA), que proporciona un proceso estructurado para la mejora continua.

Tal y como sucede con la implantación de sistemas de calidad, existen diferentes grados de desarrollo de los SGMA y diferentes alternativas para su implantación. Así los expertos en la materia señalan que una empresa deberá valorar y decidir si lo que quiere es un SGMA informal o no referenciado, no auditable y no certificable, planificado y estructurado ad hoc en función de las características de la empresa; o si bien por el contrario, necesita un SGMA formal, auditable por terceros y certificable, que tome como referencia algún estándar internacional.

En síntesis, se puede afirmar que las empresas de nuestro ámbito han utilizado principalmente dos alternativas para certificar un SGMA: la norma internacional ISO 14000 o el reglamento (CE) núm. 761/2001, del parlamento Europeo y del Consejo de 19 de marzo de 2001, de Ecogestión y Ecoauditoría, más conocido por las siglas EMAS (EcoManagement and AUdit Sheme), y que sustituye al anterior Reglamento (CEE) núm. 1836/1993. Estos dos modelos ofrecen un marco sistemático para incorporar los aspectos medioambientales en el día a día de la empresa. En la actualidad numerosas empresas españolas se han planteado ya el reto de la implantación de este tipo de estándares o se encuentran actualmente en el proceso de certificación mientras que otras tantas ya han obtenido el preciado certificado.

La norma ISO 14000

Estas normas, promulgadas por primera vez en el año 1996, tienen su origen en la cumbre de Río de 1992, a la que la ISO acudió como organismo invitado. En aquella conferencia mundial sobre el medioambiente la ISO se comprometió a crear unas normas ambientales internacionales. La ISO 14000 tomó como base la norma BS7750 publicada en 1994.

La norma ISO 14000, como ocurría con la norma ISO 9000, no es una sola norma, sino que forma parte de una familia de normas que se refieren a la gestión ambiental aplicada a la empresa, cuyo objetivo consiste en la formalización y sistematización de aquellos procesos y tareas que repercuten directa o indirectamente en el medioambiente.

Se debe dejar claro que estas normas no fijan unas metas ambientales, unos resultados medioambientales a cumplir. Estas normas establecen unos requisitos sobre la sistemática de trabajo a cumplir en la empresa respecto a las actividades que están relacionadas con el impacto ambiental.

La nueva familia de normas ISO 14000 cuenta con dos normas básicas sobre Sistemas de Gestión Ambiental: la norma ISO 14001: 2004 Requisitos con orientación para su utilización, y la norma ISO 14004: 2004 Directrices generales sobre principios, sistemas y técnicas de apoyo.

No se han realizado grandes cambios estructurales o de contenido en esta revisión, se ha realizado una unificación terminológica, se han detallado y explicado mejor alguno de los apartados y se han incorporado nuevas definiciones. La nueva versión no establece requisitos adicionales, aunque sí que recoge en la propia norma alguna de las exigencias que ya estaban siendo requeridas por los organismos certificadores en el proceso de certificación.

Tabla 5.¿Cuales han sido los cambios más importantes entre la versión de normas ISO 14001:1996 e ISO 14001:2004?

La norma ISO 14001 es la norma certificable que establece los requisitos que se han de cumplir; cuenta con los siguientes apartados:

1. Objeto y campo aplicación
2. Normas para consultar
3. Términos y definiciones.
4. Requisitos del sistema de gestión ambiental.
   - 4.1. Requisitos generales.
   - 4.2. Política ambiental.
   - 4.3. Planificación.
     - 4.3.1. Aspectos ambientales.
     - 4.3.2. Requisitos legales y otros requisitos.
     - 4.3.3. Objetivos, metas y programas.
   - 4.4. Implementación y operación.
     - 4.4.1. Recursos, funciones, responsabilidad y autoridad.
     - 4.4.2. Competencia, formación y toma de conciencia.
     - 4.4.3. Comunicación.
     - 4.4.4. Documentación.
     - 4.4.5. Control de documentos.
     - 4.4.6. Control operacional.
     - 4.4.7. Preparación y respuesta ante emergencias
   - 4.5. Verificación
     - 4.5.1. Seguimiento y medición.
     - 4.5.2. Evaluación del cumplimiento legal.
     - 4.5.3 No conformidad, acción correctiva y acción preventiva.
     - 4.5.4. Control de los requisitos.
     - 4.5.1. Auditoría interna.
   - 4.6. Revisión por la Dirección

Figura 3.ISO 14001, norma certificable

2.10 Conclusiones

- La calidad es un factor clave para competir en el duro mercado de exclusión y el mejor juez para dictaminar si los esfuerzos de las empresas para desarrollar y fabricar productos han alcanzado el éxito.

- Se consigue un éxito duradero cuando, por una parte, la estructuración de los negocios y procesos sea adecuada, robusta y transparente.

- Con la dura competencia mundial, en el futuro ganarán cuota de mercado las empresas que dominen y practiquen las reglas de calidad total, ajustando todas sus estructuras y métodos.

Capítulo 3

3 Modelos

3.1 Modelo CMM

El modelo CMM capacidad de madurez, también denominado CMM-SW por el Software Engineering Institute (SEI) como marco de referencia para la evaluación y mejora de procesos software.

CMM contiene los elementos esenciales para conseguir procesos eficaces en uno o más cuerpos de conocimiento, estando estos elementos basados en los conceptos desarrollados por los padres de la calidad: Crosby, Deming, Juran y Humphrey.

El CMM-SW fue actualizado hacia el modelo, integración del modelo de madurez de capacidades (CMMI).

Niveles de madurez

El modelo CMM organiza la madurez de un proceso software en 5 niveles.

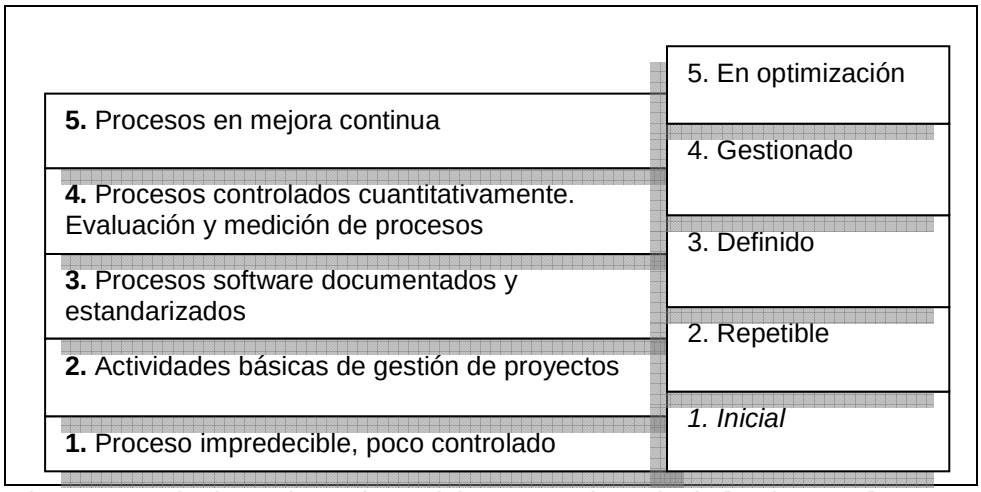


Figura 4.Escala de madurez de modelo CMM. Adaptado de [Javier Tuya]

En cada nivel, además de establecer una escala de medida de la capacidad de los procesos, se fijan unos objetivos que ayudan a la organización a priorizar los esfuerzos dedicados a la mejora de estos procesos.

Nivel	Descripción
Nivel 1 o Inicial	No existen áreas de procesos, los procesos no están definidos aunque algunos de ellos se realizan de forma controlada. El trabajo se realiza de manera impredecible. El éxito depende de las capacidades y los esfuerzos individuales realizado por las personas.
Nivel 2 o Repetible	Se encuentran establecidas las actividades básicas de gestión de proyectos, de forma que pueden observarse y reproducirse las prácticas que han tenido éxito en proyectos anteriores.
Nivel 3 o Definido	Se documentan y estandarizan tanto los procesos de desarrollo y mantenimiento de software, como los de gestión. Todos los proyectos usan una versión de los procesos integrada en la organización
Nivel 4 o Gestionado	Se establece un programa amplio y detallado de medidas, tanto para el proceso como para el producto software. Se recogen y analizan los datos de todos los proyectos, formando una base de datos cuantitativa que será de gran ayuda en la evaluación y estimación de proyectos.
Nivel 5 o En optimización	La organización está inmersa en un proceso de mejora continua de todos sus procesos, recogiendo datos de todos sus proyectos y utilizándolos para aportar nuevas ideas y para proporcionar innovaciones que utilizan nuevos métodos y tecnologías.

Tabla 6.Objetivos en cada nivel de madurez de un proceso

La arquitectura del modelo CMM se muestra en la siguiente figura:

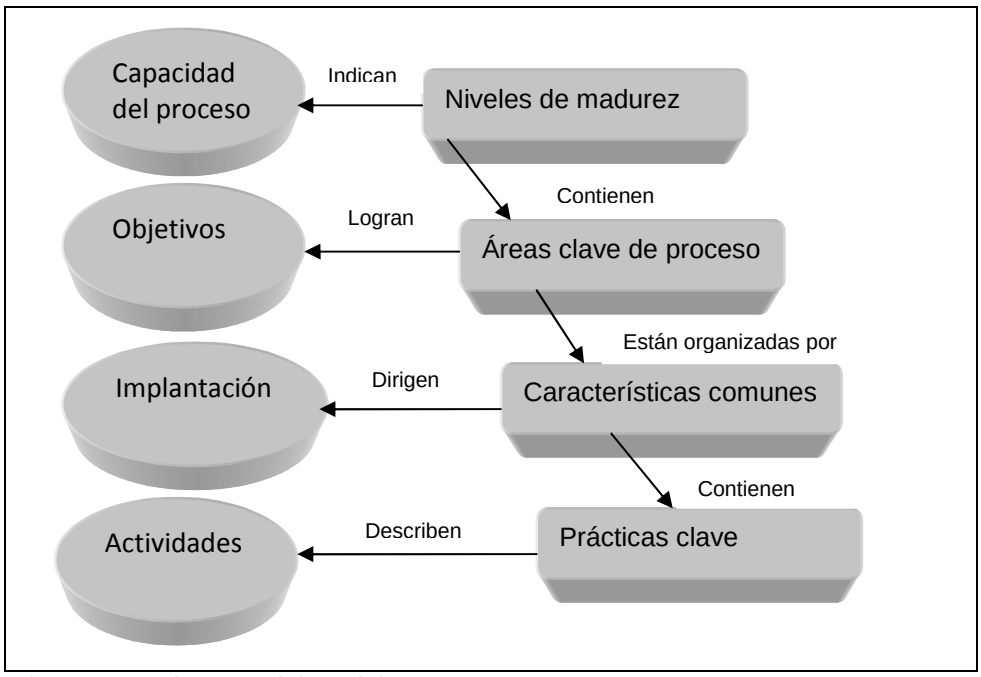


Figura 5.Arquitectura del modelo CMM

Áreas clave de proceso

Cada nivel de madurez está organizado en áreas clave de procesos. Las áreas clave de procesos representan un grupo de prácticas o actividades relacionadas que colectivamente ayudan a alcanzar un conjunto de objetivos que permiten mejorar la capacidad o madurez del proceso.

Nivel	Áreas clave de procesos
1. Inicial	No hay áreas clave de procesos establecidas.
2. Repetible	- Gestión de la configuración del software. - Aseguramiento de la calidad del software. - Gestión de acuerdos y contratos con proveedores. - Planificación, seguimiento y control de proyectos. - Gestión de requisitos.
3. Definido	- Revisión detallada de procesos. - Coordinación dentro del grupo de trabajo. - Ingeniería del producto software. - Gestión integrada del proyecto. - Programa de formación. - Definición del proceso organizativo. - Enfoque hacia los procesos organizativos.
4. Gestionado	- Gestión de la calidad de software. - Gestión cuantitativa de procesos.
5. En optimización	- Gestión del cambio en los procesos. - Gestión de los cambios tecnológicos.

Tabla 7.Relación entre los niveles de madurez y las áreas clave de procesos en el modelo CMM

Las áreas clave de procesos están a su vez organizadas en características comunes, son atributos que muestran si la implantación e institucionalización de las áreas clave de procesos ha sido efectiva, repetible y perdurable, es decir, si se han podido cumplir los objetivos de las áreas clave de procesos.

Característica	Descripción
Compromiso	Es el conjunto de acciones que la organización debe realizar para poder asegurar que el proceso es repetible y duradero. Normalmente está relacionado con las políticas de la organización y el liderazgo de la dirección.
Capacidad	Describe las precondiciones que deben darse en un proyecto o en la organización para implantar de forma efectiva los procesos de software. Habitualmente afecta a los recursos, a la estructura y a la formación
Actividades	Describe los roles y los procedimientos necesarios para implantar las áreas clave de procesos. Habitualmente incluyen procedimientos relacionados con la planificación y el seguimiento del trabajo, así como las acciones correctivas necesarias.
Medidas y análisis	Describen la necesidad de realizar mediciones de los procesos y analizan dichas medidas. Suelen incluir ejemplos de medidas tomadas para determinar el estado y la eficacia de las actividades realizadas.
Verificación	Describe los pasos que deben llevarse a cabo para asegurar que las actividades se realizan según los procesos establecidos. Habitualmente incluye revisiones y auditorías por parte de la dirección y del grupo de aseguramiento de la calidad.

Tabla 8.Características comunes de las áreas clave de procesos.

CMMI

Desde el año 1991, el modelo CMM se fue adaptando a múltiples disciplinas: Ingeniería de sistemas, Ingeniería del software, compras, desarrollo de procesos y productos, etc., derivando en modelos diferentes.

Las organizaciones que desean mejorar sus procesos en todas estas disciplinas se encontraban con que el modelo presentaba grandes diferencias de arquitectura, enfoque, contenido y aplicación. Este hecho provocaba un gran incremento del coste de la implantación de CMM en términos de formación, evaluaciones y actividades de mejora, ya que no existía una integración de todos estos modelos. El CMMI surgió como solución a todos estos problemas de falta de integración y uso de múltiples modelos CMM.

El modelo CMMI es un marco creado a partir de todos estos modelos mencionados. Su desarrollo ha sido realizado para que pueda adaptarse a múltiples disciplinas.

CMMI contiene un conjunto de productos que, además de numerosos modelos adaptables a las diferentes áreas de conocimiento, contiene métodos de evaluación según cada modelo así como material de información.

El objetivo inicial de CMM: "Obtener productos de calidad dentro de los márgenes temporales previstos con el mínimo coste" no ha cambiado en CMMI.

CMMI, igual que CMM, proporciona un enfoque disciplinado para mejorar los procesos de una organización, ayudando a establecer los objetivos de mejora y las prioridades, proporcionando guías para implantar procesos de calidad así como un marco de referencia para la realización de las evaluaciones.

En cambio, CMMI basa la aplicación de todos los principios de CMM a lo largo de todo el ciclo de vida de ingeniería, no únicamente al ciclo de vida del desarrollo del producto software.

Diferencias entre CMM y CMMI

CMMI puede ser considerado como una extensión del CMM-SW. La similitud entre ambos modelos es muy grande y las principales diferencias son una consecuencia directa de la actualización de CMM hacia CMMI.

- Se han añadido nuevas áreas de proceso. - Se han añadido mejores y más modernas prácticas clave. - Se ha añadido un objetivo genérico para cada área de proceso.

Tabla 9.Diferencias principales entre CMM y CMMI

Si se analizan estas diferencias en función del nivel de madurez en que se hallan, se pueden encontrar las siguientes áreas de proceso en el modelo CMMI que no se pueden encontrar en el modelo CMM.

Nivel	Descripción
Nivel 2	Medición y análisis. Han sido aisladas de CMM todas las prácticas relacionadas con este objetivo y han sido agrupadas dentro de esta nueva área de proceso.
Nivel 3	El área Ingeniería del producto software de CMM ha sido reemplazada en CMMI por múltiples y más detalladas áreas de proceso: Desarrollo de requisitos, soluciones técnicas, integración del producto, verificación y validación. En el área de gestión integrada del proyecto CMM se contempla la gestión de riesgos, pero ahora ha sido considerada como un área de proceso independiente. Finalmente, a este nivel se ha añadido una nueva área denominada análisis de decisiones y resolución, que no se encontraba en CMM.
Nivel 4	Este nivel ha sufrido una reestructuración y las áreas de gestión cuantitativa de procesos y gestión de la calidad de software han sido convertidas a gestión cuantitativa del proyecto y rendimiento o realización del proceso organizacional, respectivamente.
Nivel 5	No ha habido grandes cambios en este nivel, simplemente una fusión de las áreas gestión de los cambios tecnológicos y gestión del cambio en los procesos en una única área de proceso: Innovación organizacional y despliegue. El área de prevención de defectos ha sido reestructurada y renombrada a análisis causal y resolución.

Tabla 10. Áreas de proceso en el modelo CMMI

Proceso personal del Software (PSP) y Proceso del equipo Software (TSP)

Proceso personal del Software (PSP)

Es un modelo de mejora del proceso software formado por un conjunto estructurado de descripciones de procesos, de mediciones y de métodos, basado en la aplicación de métodos avanzados y tradicionales de ingeniería al desarrollo de software y orientado a la mejora individual de cada ingeniero de software.

Fue desarrollado por **Watts Humphrey (Humphrey, 1997a),** quien había participado en la creación de la primera versión del modelo CMM en la década de los 80 y del que algunos decían que no era aplicable a pequeñas organizaciones y a proyectos pequeños. Para demostrar que estas afirmaciones eran erróneas, Watts Humphrey desarrolló, entre los años 1989 y 1993, más de 60 programas y escribió más de 25000 líneas de código, aplicando los principios de CMM hasta el nivel 5.

Concluyó que los principios incluidos en el modelo CMM eran aplicables incluso en el trabajo de un único ingeniero de software y de ahí surgió el modelo PSP.

Según PSP, para realizar un buen trabajo de ingeniería de software, un técnico debe, en primer lugar, conocer el tiempo que necesita para realizar bien su

trabajo, en segundo lugar, planificarlo antes de comenzarlo, y en tercer lugar realizarlo de forma correcta. Finalmente, se deberán analizar los resultados de cada actividad y utilizarlos para mejorar los procesos, actividades y tareas.

- Cada técnico es diferente. Para ser efectivos deben tener en cuenta sus capacidades personales y adaptar la planificación de sus trabajos a sus tiempos. - Las personas que trabajan en el desarrollo de software pueden mejorar su rendimiento utilizando procesos bien definidos. Para conocer el rendimiento de cada persona es necesario medir el tiempo o esfuerzo dedicado a cada actividad o tarea. Los defectos que se generan y que se corrigen durante ese periodo de tiempo y los tamaños de los productos que se obtienen. Es decir, el PSP utiliza tres tipos de mediciones: esfuerzo, número de defectos y tamaño de los productos. - Para obtener productos de calidad, cada persona debe responsabilizarse del trabajo que ha realizado y de los productos que ha obtenido. - Para reducir los costes de producción, se debe intentar prevenir los defectos antes de que ocurran y en todo caso, descubrir los defectos en las fases más tempranas. - Mejorar la gestión de la calidad de un proyecto. - Mejorar la estimación y la planificación de un proyecto. - Reducir el número de defectos en los productos obtenidos durante el desarrollo.

Tabla 11.Principios del PSP

La arquitectura principal del modelo PSP:

1. Durante la fase 0 el objetivo principal del ingeniero de software es aprender a seguir un proceso definido y a recopilar los datos básicos sobre tamaño, tiempo y defectos.

2. Durante la fase 1, se dispone de datos históricos para el proceso, el objetivo principal se centra en la estimación y en la planificación. Para ello es necesario conocer métodos de estimación de esfuerzo y tamaño, y utilizar estos resultados en la planificación y seguimiento.

3. Durante la fase 2, debido a que se realiza el control de la planificación, el objetivo principal está en la gestión de calidad. El técnico debe conocer los métodos de detección temprana y de eliminación de defectos.

Una vez que se ha finalizado con las tres fases, el técnico debe realizar un informe sobre su rendimiento y debe ser capaz de analizar los datos que ha ido recopilando para efectuar los cambios que considere oportunos y que le conduzcan a la mejora de su trabajo.

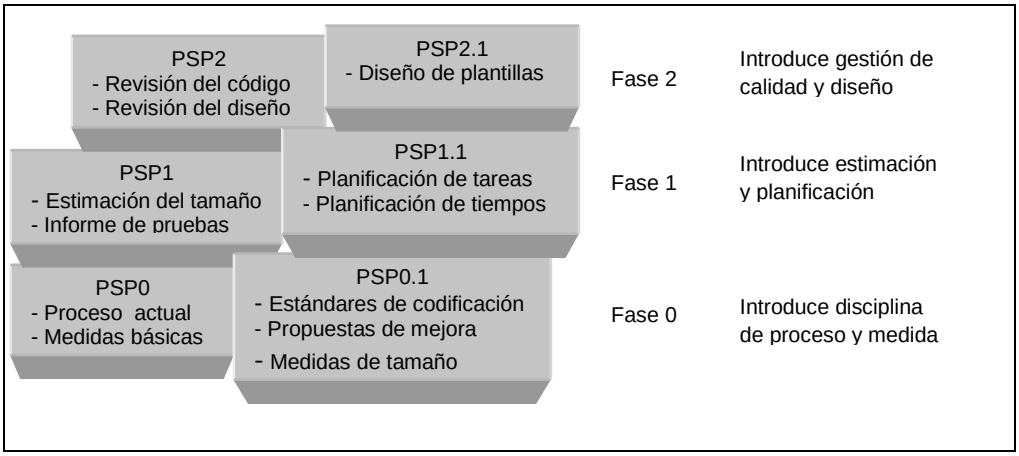


Figura 6.Arquitectura del modelo PSP. Adaptado de [Javier Tuya]

En el modelo PSP se definen las siguientes actividades:

Actividad	Descripción
Planeación	Aísla los requisitos y realiza estimaciones tanto del tamaño como de los recursos. Desarrolla la estimación de los defectos, e identifica las tareas de desarrollo y se crea un programa para el proyecto.
Diseño de alto nivel	Se realizan especificaciones externas para cada uno de los componentes que se van a construir y por consiguiente se crea el diseño de componentes. Si hay incertidumbre se realizan prototipos. Se registran aspectos relevantes y por ende se le da seguimiento.
Revisión de alto nivel	Se aplican métodos de verificación formal para descubrir errores en el diseño.
Desarrollo	Se mejora y revisa el diseño del componente. El código se genera, revisa, compila y prueba.
Post mórtem	Se determina la eficacia del proceso por medio de medidas y mediciones obtenidas. Las medidas y mediciones deben dar la guía para modificar el proceso a fin de mejorar su eficacia.

Tabla 12.Actividades del modelo PSP

Proceso del equipo Software (TSP)

Proceso del equipo Software (TSP) es un método de establecimiento y mejora del trabajo en equipo para procesos software. TSP proporciona directrices para ayudar a un equipo a establecer sus objetivos, a planificar sus procesos y a revisar su trabajo con el fin de que la organización pueda establecer prácticas de ingeniería avanzada y así obtener productos eficientes, fiables y de calidad (Humphrey). Un objetivo de TSP es proporcionar al equipo un entorno que soporte el trabajo según establece PSP.

- Los técnicos conocen muchas cosas sobre su trabajo y pueden realizar las mejores planificaciones. Cuando son ellos quienes planifican su propio trabajo, se encuentran comprometidos con el plan. - Un seguimiento preciso de un proyecto requiere planes bien detallados y datos precisos. Únicamente el personal que realiza el trabajo es capaz de recoger con precisión dichos datos. - Para minimizar el tiempo del proyecto, los ingenieros deben equilibrar su carga de trabajo. - Para maximizar la productividad, el primer foco de atención debe ser la calidad.

Tabla 13.Principios de TSP

TSP está formado por dos componentes primarios bien diferenciados que abarcan distintos aspectos del trabajo en equipo y que pueden observarse de manera resumida.

a) Formación del equipo de trabajo.
b) Gestión del equipo de trabajo.

Humphrey define los siguientes objetivos:

- Formar grupos autodirigidos que planeen y den seguimiento a su trabajo, establezcan metas y que sean propietarios de sus procesos y planes. - Exponer a los directivos como dirigir y estimular a sus equipos y como ayudarles a mantener un rendimiento máximo. - Apresurar la mejora del proceso del software, haciendo el modelo de madurez de la capacidad, CMM, nivel 5, el comportamiento habitual y esperado. - Brindar a las distintas organizaciones una guía para la mejora. - Facilitar la enseñanza universitaria de aptitudes de equipo con grado industrial.

Tabla 14.Objetivos para el TSP

Un equipo autodirigido tiene:

a) Comprensión de las metas y objetivos, define el rol y responsabilidad de cada persona del equipo.

b) Da seguimiento cuantitativo a los datos del proyecto, es decir, productividad y calidad.

c) Identifica un proceso de equipo que sea adecuado para el proyecto y una táctica para implementarlo.

d) Evalúa de forma continua el riesgo y reacciona en consecuencia.

e) Da seguimiento, administra e informa del estado del proyecto.

El TSP define las siguientes actividades:

a) Inicio del proyecto.
b) Diseño de alto nivel.
c) Implementación.
d) Integración y pruebas.

e) Post mórtem.

Estas actividades permiten que el equipo planee, diseñe y construya software de manera ordenada, de la misma manera mide cuantitativamente el proceso y el producto. La etapa post mórtem es el escenario de las mejoras del proceso.

Relación entre PSP, TSP Y CMM

a) Para que una organización mejore el proceso software y desarrolle productos de calidad, debe contemplar diferentes aproximaciones de mejora del proceso, todas ellas complementarias.

b) La organización debe llevar sus procesos según la forma establecida en un modelo. Humphrey propone utilizar el modelo CMM.

c) Los ingenieros deben desarrollar sus capacidades individuales de una forma eficiente y controlada. Es conveniente utilizar los principios PSP.

d) El trabajo debe desarrollarse dentro de un equipo formado y con una capacidad de trabajo conjunta. Es conveniente utilizar los principios TSP.

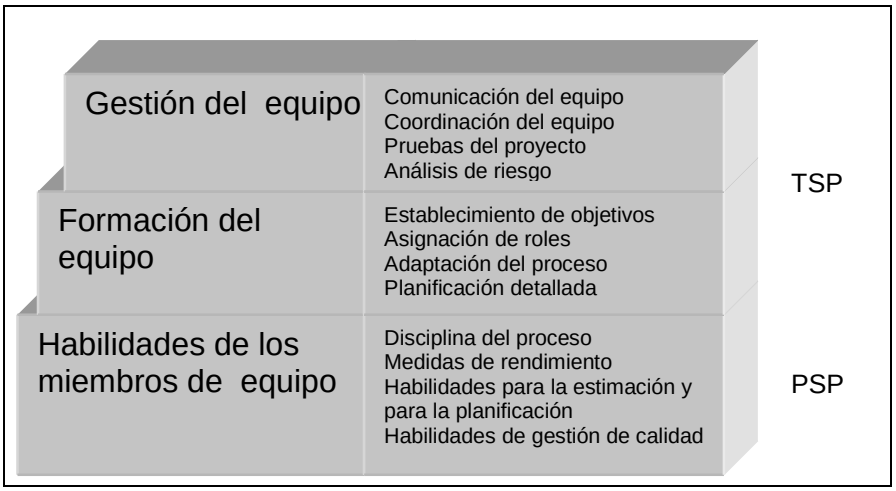


Figura 7.Relación entre PSP y TSP. Adaptado de [Javier Tuya]

La figura 7 muestra de manera conjunta PSP y TSP, permite definir y mantener equipos que consideren las capacidades individuales de sus miembros, es decir, si bien esta capacidad individual es crítica, ya que cada instrucción de un módulo ha sido realizada por un único ingeniero de software, también es cierto que un producto software es el resultado de un trabajo en equipo, formado por un conjunto de subproductos o módulos que han sido diseñados, construidos, integrados, probados y mantenidos por un equipo de ingenieros de software cuyas habilidades, capacidades y disciplina, contribuirán en el éxito del proyecto.

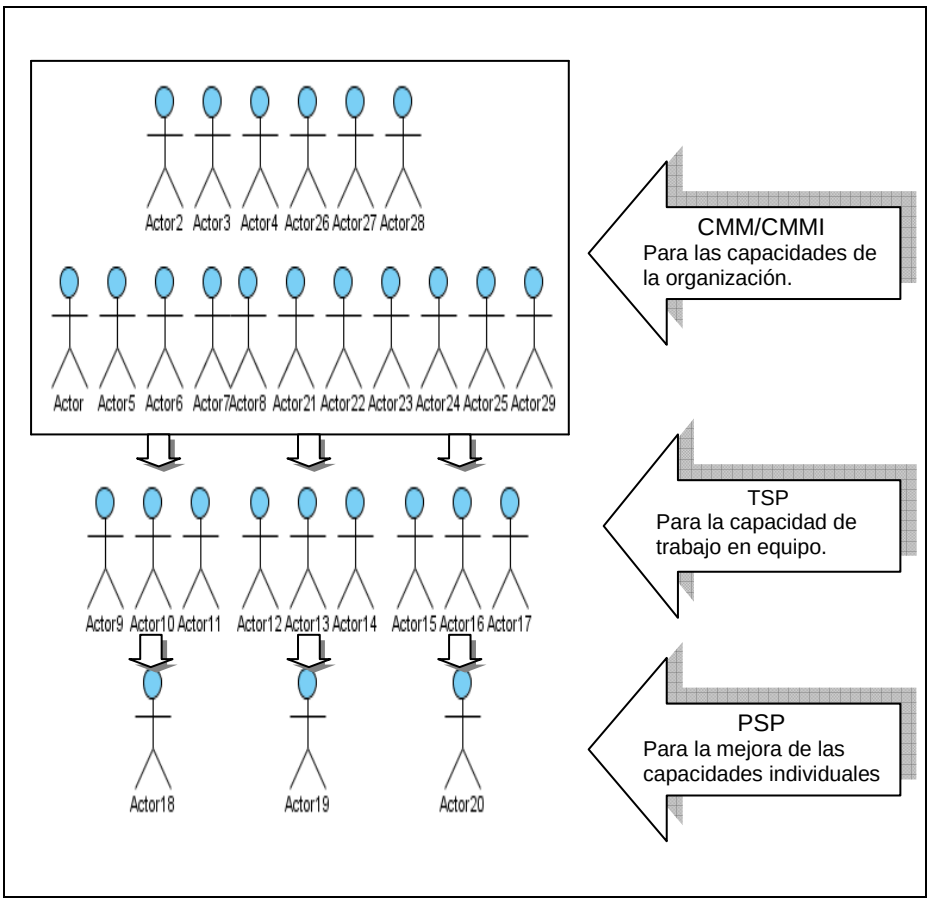


Figura 8.Relación entre CMM/CMMI, TSP y PSP. Adaptado de [Javier Tuya]

En la figura 8 se puede observar como CMM/CMMI, TSP y PSP puede usarse de forma conjunta y combinada para mejorar las capacidades de toda la organización.

PSP forma técnicos de equipos establecidos con TSP en la mayoría de las prácticas genéricas de CMM/CMMI. TSP por sí solo, aunque se ha aplicado a todos los equipos de desarrollo, no cubre todas las prácticas de cada área de proceso CMM/CMMI, razón de más para que sea utilizado de forma complementaria a este modelo, no de forma aislada.

Debido a que las actividades de medición y análisis de los resultados son fundamentales en PSP y en TSP, su utilización durante la aplicación de CMM/CMMI en una organización, permite acelerar el progreso y aumentar el nivel de capacidad de la empresa en un tiempo menor que sin su uso. El uso combinado de TSP y PSP acelera el proceso de mejora según CMM.

- En primer lugar, con la aplicación de TSP, los técnicos y los equipos, pueden ver las razones del alto nivel de madurez de la mayoría de prácticas de CMM, y entienden más la necesidad de cooperar en los esfuerzos de mejora.
- En segundo lugar, ya que el objetivo de cualquier esfuerzo de mejora de procesos de software es aumentar el rendimiento de la organización, y ello requiere cambios en el funcionamiento de los procesos de ingeniería, cualquier esfuerzo de mejora debe estar acompañado por pasos que pueden demostrar cambios en los comportamientos de ingeniería. TSP y PSP se utilizan para este fin.
- En tercer lugar, cualquier esfuerzo de mejora conlleva el riesgo de burocratizarse y de imponer presión sobre los ingenieros en lugar de ayudarles. Si como se sugiere en TSP, el equipo es tratado como si fueran clientes, este riesgo disminuye considerablemente.
- En cuarto lugar, TSP permite mejorar substancialmente el rendimiento de los grupos de software en una organización.

Tabla 15.Establecimiento de pasos para la combinación de ambos esfuerzos

En el informe se detallan las actuaciones que se deben aplicar según los principios de TSP en cada nivel de madurez del modelo CMM y para cada área clave de proceso. Humphrey afirma que, contrariamente a la impresión errónea que tienen algunas personas, TSP y PSP pueden ser aplicados en una empresa obteniendo grandes beneficios, independientemente del nivel de madurez en que ésta se encuentre.

3.2 El modelo SPICE

El proyecto SPICE (Software Process Improvement and Capability Determination: Determinación del mejoramiento y capacidad del proceso de software), representa el mayor marco de colaboración internacional establecido con la finalidad de desarrollar un estándar de evaluación de procesos de software. Se inició con tres objetivos claves:

- Desarrollar un marco de trabajo común para la evaluación y mejora de procesos de software.
- Aplicar el estándar desarrollado en la industria del software.
- Promover la transferencia de conocimiento y de tecnología sobre procesos de software entre todas las empresas.

Tabla 16.Objetivos del modelo SPICE

Proporciona un marco conceptual de valoración que cumple con ISO/IEC 15504: 2003 e ISO/IEC 12207.

ISO/IEC 15504, es un estándar internacional que es aplicable a cualquier organización/empresa que quiere conocer y mejorar la capacidad de sus procesos, independientemente del tipo de organización, del modelo del ciclo de vida adoptado, de la metodología de desarrollo y de la tecnología utilizada.

Proporciona una base para realizar evaluaciones de la capacidad de los procesos de software y permite reflejar los resultados obtenidos sobre una escala común, que puede usarse, por una parte, para comprobar la evolución

de una organización en el tiempo o para observar su situación respecto a la competencia y, por la otra, para la definición de estrategias de mejora.

El estándar completo vigente hoy en día se denomina ISO/IEC 15504, Information Technology-Process Assessment: Evaluación de Procesos de la tecnología de la información.

- ISO/IEC 15504-1:2004. Parte 1: Conceptos y vocabulario. Representa una introducción de la norma, proporciona un guía de uso de la misma. Se incluyen términos definidos específicamente para la norma. - ISO/IEC 15504-2:2003/Cor 1:2004. Parte 2: Desarrollo de la evaluación. Define los requisitos que debe cumplir una evaluación para que produzca resultados repetibles, fiables y consistentes. - ISO/IEC 15504-3:2004. Parte 3: Orientación sobre cómo realizar una evaluación. Proporciona una guía para poder utilizar los resultados de una evaluación en la mejora de los procesos evaluados. - ISO/IEC 15504-5. Parte 5. Ejemplar del proceso de evaluación del modelo. Proporciona un modelo totalmente compatible con la parte normativa, incluye un conjunto de indicadores que facilitan el cálculo de la capacidad de los procesos.

Tabla 17.Partes en que se divide ISO/IEC 15504

La dimensión de la capacidad define una escala de valoración para la capacidad de los procesos que consta de 6 niveles, desde el 0 hasta el 5.

Escala de valoración	Descripción
Nivel 0. Incompleto	El proceso no existe o no se consigue su propósito.
Nivel 1. Realizado	Se alcanza el propósito del proceso en términos generales. El personal de la organización reconoce que el proceso se realiza cuando es necesario, pero no se hace de una forma planificada ni se realiza ningún seguimiento.
Nivel 2. Gestionado	Se obtienen los productos del proceso, de acuerdo con una planificación y realizándose un seguimiento. La principal diferencia con el nivel 1, es que se generan productos que cumplen completamente con los requisitos de calidad y lo hacen dentro de los plazos de tiempo y con los recursos establecidos.
Nivel 3. Establecido	El proceso se realiza y se gestiona utilizando procedimientos definidos según los principios de la ingeniería del software. La principal diferencia con el nivel 2, es que se usa un proceso definido y con capacidad para alcanzar los resultados esperados.
Nivel 4. Predecible	La realización del proceso se gestiona de forma cuantitativa, es decir, se recogen medidas detalladas del nivel de realización del proceso y se analizan. La principal diferencia con el nivel 3 es que ahora el proceso se lleva a término de manera consistente dentro de unos límites predefinidos.
	La realización de un proceso se optimiza de forma continuada, de cara a su contribución de alcanzar los objetivos de negocio de la organización. Se establecen objetivos cuantitativos de eficacia y eficiencia en la realización de los procesos, basados en los objetivos de

Nivel 5. En optimización	negocio de la organización. Se lleva a cabo una monitorización continua de los procesos y se analizan los datos obtenidos. Esto permite que los procesos estándar definidos dentro de la organización cambien dinámicamente, para adaptarse de forma efectiva a los actuales y futuros objetivos de la empresa. La principal diferencia con el nivel 4, es que ahora los procesos, definidos y estandarizados, cambien de manera dinámica, y se adapten para satisfacer con eficacia los objetivos actuales y futuros del negocio.

Tabla 18.Dimensión de la capacidad de ISO/IEC 15504

De ISO/IEC 12207 se obtienen todos los procesos que una organización puede realizar para comprar, suministrar, desarrollar, operar, mantener y dar soportar software. La dimensión de la capacidad, formada por seis niveles de capacidad y nueve atributos de procesos, proporciona una base para medir la capacidad de dichos procesos, en función del grado de cumplimiento de sus atributos.

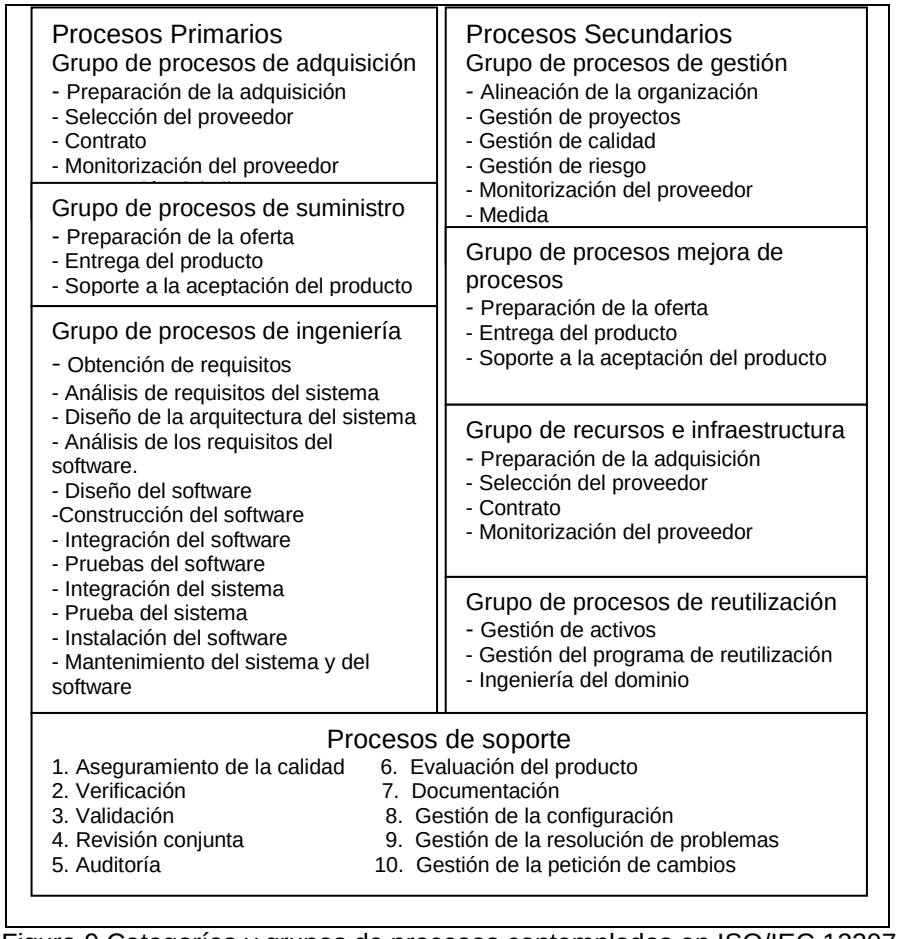


Figura 9.Categorías y grupos de procesos contemplados en ISO/IEC 12207. Adaptado de [Javier Tuya]

3.3 Conclusiones

- La orientación a procesos en el desarrollo del software y la necesidad de mejora de las actividades y tareas que se realizan durante estos procesos, han propiciado la investigación en este campo y la aparición de algunos modelos de evaluación y mejora de los procesos del ciclo de vida del software.

- Los modelos se venían aplicando principalmente a las grandes compañías, desde hace unos años ha crecido el interés en las pequeñas y medianas empresas (PYMEs). Debido a que las características de estas empresas son, muy diferentes a las grandes, se han creado modelos específicos para las PYMEs y se han realizado adaptaciones de los modelos más aplicados, CMM y SPICE a este tipo de empresas.

- La adherencia a un modelo de procesos en una empresa puede suponer tanto una mejora interna de la misma, como un reconocimiento que ofrezca una ventaja competitiva para la adjudicación de nuevos proyectos. Ambos factores son cada vez más tenidos en cuenta en el sector del desarrollo de software y se está produciendo actualmente un aumento notable del interés por estas buenas prácticas.

Capítulo 4

4 Estándares

4.1 Plan de aseguramiento de la calidad IEEE Std 730-2002

4.1.1 Visión

4.1.1.1 Ámbito de aplicación

Esta norma se aplica al desarrollo de un plan de aseguramiento de la calidad del software (PACS) que podemos definir como:

"Un conjunto, sistemático y planificado, de acciones necesarias para proveer la evidencia adecuada de que el proceso de desarrollo o mantenimiento de un sistema de software cumple los requerimientos técnicos funcionales tan bien como los requerimientos gerenciales para cumplir la planificación y operar dentro del presupuesto confirmados".[DANIEL GALIN]

La garantía de calidad o aseguramiento de la calidad consiste en la auditoría y las funciones de información de la gestión. El objetivo de la garantía de la calidad es proporcionar la gestión para informar de los datos necesarios sobre la calidad del producto, por lo que se va adquiriendo una visión más profunda y segura de que la calidad del producto está cumpliendo sus objetivos.

Es de esperar que si los datos proporcionados mediante la garantía de la calidad identifican problemas, la gestión afronte los problemas y aplique los recursos necesarios para solventarlos.

4.1.1.2 Propósito

El propósito de esta norma tiene por objeto establecer los requisitos uniformes, mínimo aceptable para la preparación y el contenido de los planes de aseguramiento de la calidad del software.

Una adecuada calidad del producto es inalcanzable sin una buena organización del desarrollo. La garantía de calidad o aseguramiento de la calidad consiste en la auditoría y las funciones de información de la gestión.

El objetivo de la garantía de la calidad es proporcionar la gestión para informar de los datos necesarios sobre la calidad del producto, por lo que se va

adquiriendo una visión más profunda y segura de que la calidad del producto está cumpliendo sus objetivos. Es de esperar, que si los datos proporcionados mediante la garantía de la calidad identifican problemas, la gestión afronte los problemas y aplique los recursos necesarios para resolverlos.

4.1.1.3 La conformidad con esta norma

El contenido de conformidad con esta norma podrá invocarse a todos los requisitos (indicado por "deberá") y se llevan a cabo tal como se define en esta norma. El formato de conformidad con esta norma puede ser válido si el resultado PACS está en el formato especificado. La palabra "tiene" se utiliza para expresar una exigencia, "debe" para expresar una recomendación, y "puede" para expresar métodos alternativos u opcionales que satisface un requisito.

4.1.2 Referencias

En el capítulo 7 contiene información en “bibliografía utilizada”.

4.1.3 Definiciones y siglas

Al finalizar el capítulo 4 de la norma se encuentran las siglas con su correspondiente definición que se han utilizado en el estudio de los distintos estándares aplicados

4.1.4 Plan de aseguramiento de calidad del software

El PACS incluye las secciones que figuran a continuación, se ajustan en el formato de esta norma. Las secciones PACS incluyen los siguientes:

1.1 Objetivo.
1.2 Documentos de referencia.
1.3 Gestión.
1.4 Documentación.
1.5 Las normas, prácticas, convenciones y métricas.
1.6 Software comentarios.
1.7 Prueba.
1.8 El problema de información y medidas correctivas.
1.9 Herramientas, técnicas y metodologías.
1.10 El control de los medios de comunicación.
1.11 Control de proveedores.
1.12 Documentos de recogida, mantenimiento y conservación.
1.13 Formación.
1.14 Gestión del riesgo.

A partir del plan de gestión, se genera el plan de aseguramiento de la calidad del software, realizaremos una breve descripción del plan de gestión del proyecto software el cual debe incluir y describir lo siguiente:

- El día a día del proyecto, con los correspondientes controles de auditoría y revisiones.
- Planificación del aseguramiento de la calidad del software.
- Planificación de la documentación del proyecto.

El plan de aseguramiento de la calidad del software es específico para cada proyecto, siguiendo las directrices del manual de calidad y de sus procedimientos y las normas requeridas por los clientes, el PACS debe incluir:

4.1.4.1 Objetivo

Se delinean los objetivos específicos y el alcance de lo particular PACS. Se deberá indicar la parte del ciclo de vida del software cubierto por el PACS para cada artículo del software especificado.

El plan de aseguramiento de la calidad del software debe contemplar los siguientes objetivos tales como:

- Facilitar el aseguramiento de políticas, procedimientos, estándares, normas para desarrollo de las aplicaciones del software.
- Asegurar que el desarrollo de las aplicaciones software se mantengan y modifiquen, así mismo las actividades que se realizan de una manera efectiva y eficiente.
- Organización del equipo de personal, así como la dirección y seguimiento del desarrollo.
- Modelo del ciclo de vida especificando sus fases respectivas tales como actividades y tareas.
- Documentación requerida, especificando el contenido de cada documento. Se definirá la documentación que se utilizará para asegurar la calidad, se generan los siguientes documentos:

    - Plan de aseguramiento de la calidad.
    - Plan de configuración del software.
    - Plan de administración del proyecto software.
    - Especificación de requisitos software.
    - Documentos de diseño software.
    - Documentos de pruebas de software.
    - Manual de usuario.
    - Plan de mantenimiento.

- Revisiones y auditorías que se realizarán durante el desarrollo del ciclo de vida del software, para garantizar que son correctos, completos y consistentes.
- Pruebas que se realizan en los distintos niveles sobre el producto del software.

- Etapa de mantenimiento describiendo como se desarrollaran los cambios sobre el producto ya en desarrollo.
- Alcanzar el nivel de calidad deseado.

4.1.4.2 Documentos de referencia

Define la lista de los documentos más importantes dentro del PACS.

Especificar referencias de:

- Proyecto.
- Documentación.
- Estándares.
- Información adicional.

4.1.4.3 Gestión

En esta sección describirá la estructura del proyecto de la organización sus tareas y sus funciones.

4.1.4.3.1 Organización

En esta sección se describen la estructura organizacional que influye y controla la calidad del software.

Los objetivos principales de la base de SQA de la organización son los siguientes:

- Desarrollar y apoyar la implementación de los componentes de SQA.
- Detectar las desviaciones de los procedimientos de SQA y la metodología.
- Sugerir mejoras en los componentes de SQA.
- Coste de fallos: Se derivan por la falta de calidad del producto o del servicio (internos y externos).

Un ingeniero especifico, proporcionado por la organización del aseguramiento de la calidad se designará como líder de configuración para la duración del proyecto. Cada integrante es responsable de la calidad de su trabajo. La organización para el SQA proporciona un equipo de ingenieros para realizar dicha función.

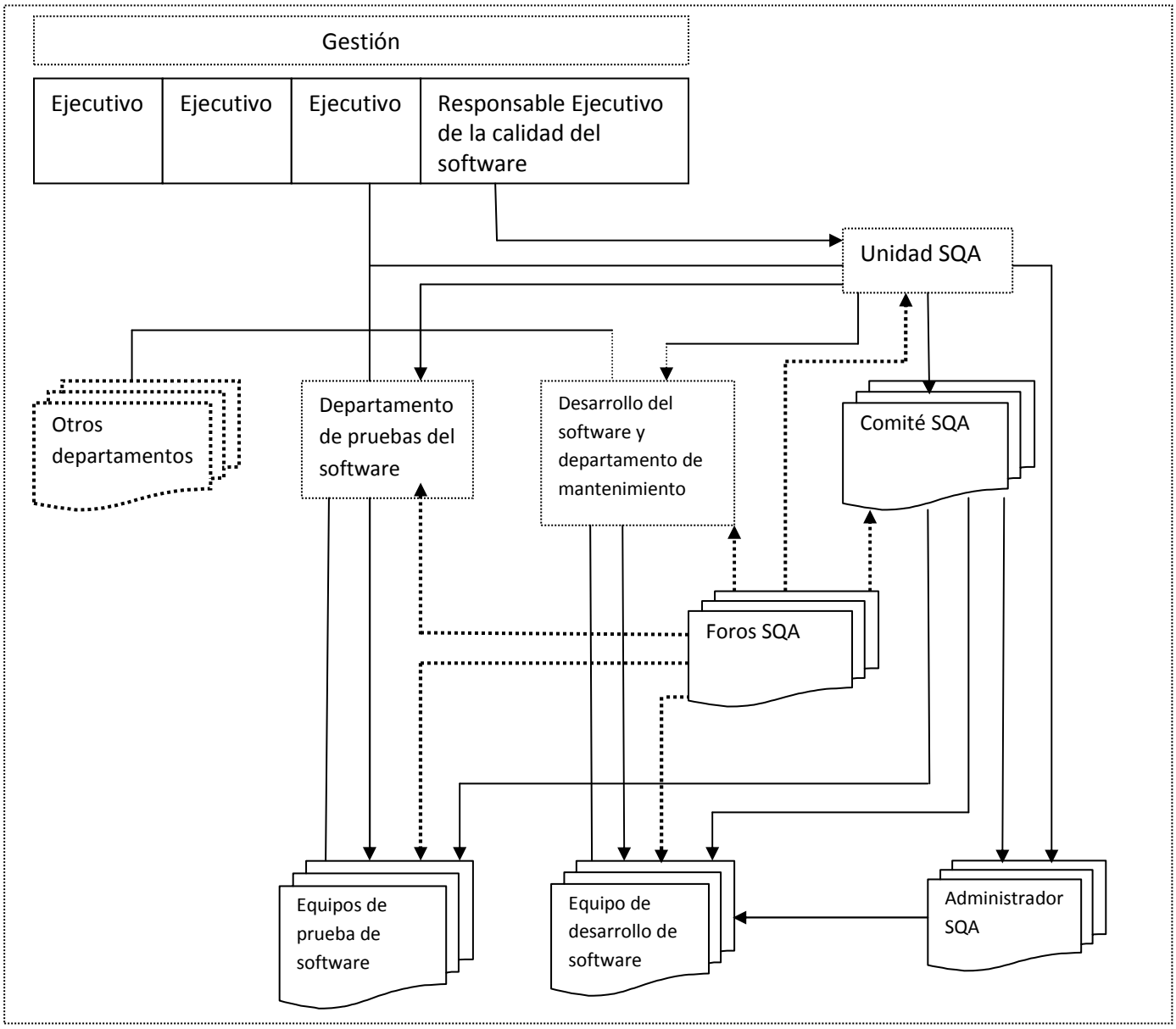


Figura 10.Organigrama de SQA

El organigrama se agrupa en tres grandes grupos:

1. Gerentes.
2. Probadores.
3. Profesionales de SQA e interesados.

1. Gerentes:

- Ejecutivos de alta dirección, en especial el ejecutivo directamente es el responsable de garantizar la calidad del software.
- Desarrollo de software y gerente de departamento de mantenimiento.
- Gerentes del departamento de pruebas del software.
- Gerentes de proyectos y jefes de equipo de proyectos de desarrollo y mantenimiento.
- Los líderes de los equipos de pruebas de software.

2. Probador:

   - Son miembros de los equipos de pruebas de software.

3. Profesionales de la SQA y profesionales interesados:

   - Miembros de la unidad del equipo SQA.
   - Administradores SQA.
   - Los miembros del comité SQA.
   - Miembros del foro SQA.

La estructuración del organigrama de la figura 10 se compone de la siguiente manera:

- Alta dirección.
- Departamento de gestión.
- Gestión de proyectos.
- Profesionales de la SQA y profesionales interesados.

4.1.4.3.2 Tareas

En esta sección se deberá describir:

a.) La parte del ciclo de vida del software cubierto por el PACS.
b.) Tareas a realizar.
c.) Criterios de entrada y salida para cada tarea.
d.) La secuencia y las relaciones de las tareas.
e.) Documentación.
f.) Reuniones de revisiones.
g.) Verificación.
h.) Actividades diseñadas para mejorar el proceso de aseguramiento de la calidad en sí.

- Alta dirección

Nombre	Tareas
Alta dirección	- Aseguramiento de la calidad de los productos, servicios y mantenimiento software. - Aseguramiento de los objetivos de calidad determinados para la organización y el sistema. - En todos los niveles, la importancia de comunicación de la calidad de los productos y servicios. - Aseguramiento del correcto funcionamiento y cumplimento de los requisitos de parte del cliente. - Aseguramiento de los recursos por el sistema de SQA. - Comienzo de la planificación y control de la realización de cambios tanto internos como externos en relación a los clientes de la organización, competencia y tecnología.

Tabla 19.Tareas de la Alta dirección

- Departamento de gestión

Nombre	Tareas
Sistema	Incluye actividades que se realizan a nivel de departamento: - Preparación de programas por cada departamento: actividades y presupuesto. Esto debe estar basado en las recomendaciones de la unidad SQA. - Preparación de planes basado en las recomendaciones de la unidad SQA. - Control de ejecución de las actividades, programas y proyectos de desarrollo. - Presentación de los temas a la alta dirección, personal ejecutivo a cargo de la calidad del software
Proyectos	Oscilan de acuerdo a los procedimientos de la organización y distribución, implican: - Control de procedimientos de la organización. - Seguimiento de los resultados del contrato y aprobación de la propuesta. - Seguimiento de pruebas de software, aprobación en el proyecto de los productos del software. - Seguimiento de la calidad de servicios de mantenimiento. - Seguimiento de los riesgos y por consiguiente solución en el proyecto. - Seguimiento del proyecto de acuerdo con los requisitos del cliente y su satisfacción.

Tabla 20.Tareas del departamento de gestión

- Gestión de proyectos

La gestión de proyectos se define en los procedimientos e instrucciones de trabajo, incluyen: ayuda de profesionales, gestión.

Nombre	Tareas
Ayuda de profesionales	- Participación conjunta entre cliente y proveedor. - Preparación de plan, calidad, actualización. - Seguimiento de asistencia, contratación, formación e instrucción de la plantilla del equipo del proyecto.
Gestión	Gestores de proyecto frente a la realización de seguimiento: - Desarrollo de actividades de revisión y correcciones. - Desarrollo de software y rendimiento con respecto al desarrollo, integración, sistema de ensayo, correcciones, pruebas de regresión. - Pruebas de aceptación. - Instalación del software. - Formación e instrucción a los miembros del equipo. - Horarios y recursos asignados. - Petición del cliente y satisfacción. - Evolución de los riesgos, aplicación de soluciones y control de resultados.

Tabla 21.Tareas de gestión de proyectos

- Profesionales de la SQA y profesionales interesados.

- Miembros de la unidad del equipo SQA.
- Administradores SQA.
- Los miembros del comité SQA.
- Miembros del foro SQA.

Cabe hacer hincapié entre los miembros antes mencionados, sólo los miembros de la unidad SQA dedican todas sus actividades en el trabajo de SQA a tiempo completo, los otros tienen responsabilidades a tiempo parcial es decir ofrecen voluntariamente su tiempo debido a su interés por la calidad.

SQA miembros de la unidad del equipo

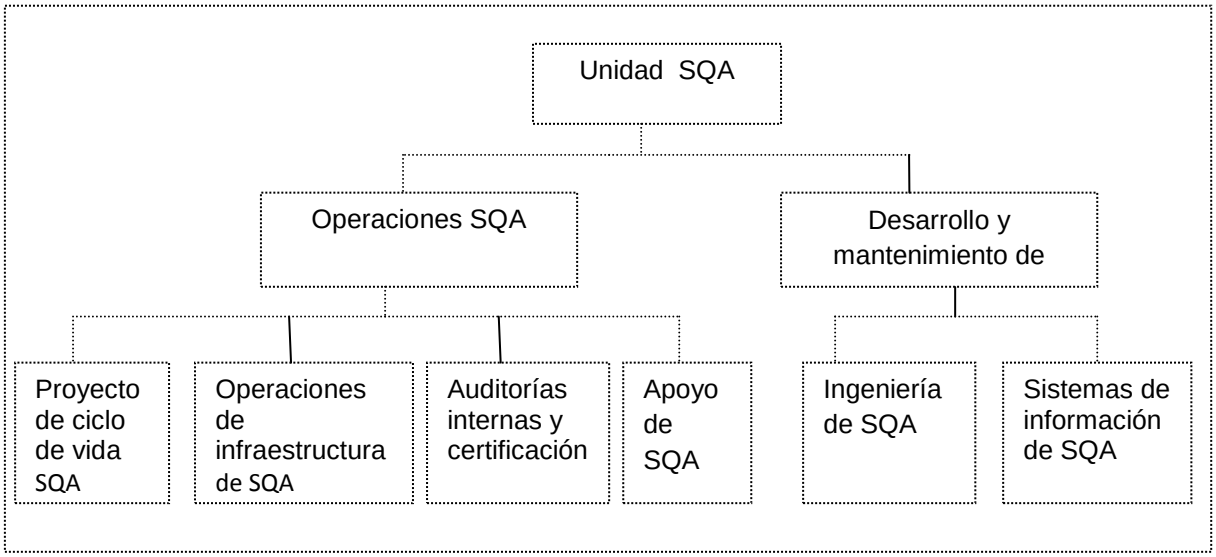


Figura 11.Esquema unidad de equipo

Cabeza o Unidad de SQA

Nombre	Tareas
Cabeza o Unidad de SQA	Realiza tareas como: - planificación, gestión de la unidad, mantener contactos con los clientes y otros organismos externos. - Realización de todas aquellas tareas de control de calidad. - Contacto con los clientes y otros organismos externos, y de igual manera con la persona a cargo de la calidad. - Elaboración de informes especiales. Ejemplo el estado de las cuestiones de la calidad del software dentro de la organización y los informes mensuales de rendimiento. - Planificación, actualización del sistema de gestión de la calidad del software en la organización. - Nombrar los miembros del equipo, los miembros del comité y administradores SQA. - Colaboración con los clientes respecto a la calidad de los productos y servicios prestados.

	- Participación en las revisiones del diseño formal. - Consulta con el jefe(s) de proyecto y jefe(s) de equipo. - Participación en comités y foros de SQA.

Tabla 22.Tareas de la unidad de SQA

Las unidades SQA se agrupan en dos grandes grupos:

1. Operaciones de SQA.
2. Desarrollo y mantenimiento de SQA.

1. Operaciones de SQA

Nombre	Tarea
Proyecto ciclo de vida	Realiza tareas como: - Revisiones ya sean de contratos, diseño y pruebas del software. - Seguimiento del cumplimiento de los equipos de desarrollo y mantenimiento. - Aprobación o recomendación de productos de software (informes de diseño y código) de acuerdo con los procedimientos establecidos. - Seguimiento de la entrega de servicios o productos de mantenimiento de software para clientes internos y externos. - Satisfacción de seguimiento de clientes (por medio de encuestas, etc.). - Los comentarios formales de diseño. - Pruebas de software, incluyendo pruebas de aceptación del cliente. - Preparación del plan de la calidad del proyecto. - Actualización de proyectos.
Operaciones de infraestructura de SQA	Realiza tareas como: - Actualización de los procedimientos, formación y seguimiento. - Apoyo a los dispositivos de calidad (plantillas, listas de control). - Documentación de control. - Acciones preventivas y correctivas. - Seguimiento de las actividades de gestión de configuración. - Capacitación del personal, formación y certificación - Seguimiento en los procedimientos e instrucciones de trabajo.
Auditorías y	Realiza tareas como: - Realización de auditorías internas SQA - Realización de auditorías SQA de los subcontratistas y proveedores. - Auditorías externas realizadas por organismo de certificación. - Las auditorías externas realizadas por los clientes que analizan el sistema de SQA antes de aceptar la organización como un proveedor. - Recogida de datos sobre el desempeño de los

certificación	subcontratistas y proveedores internos, así como de fuentes externas. - Coordinación de los contenidos de auditoría de certificación y horario. - Asegurar que las correcciones y las mejoras se lleven a cabo. - Coordinación de los contenidos de auditoría y calendario.
Apoyo de SQA	Realiza tareas como: - Proyecto del plan de calidad, elección de la metodología, aplicación de procedimientos. - Selección de metodologías y herramientas de desarrollo. - Selección de métricas de SQA y componentes de los costos de software. - Selección de las medidas para solventar los riesgos identificados en el desarrollo del software. - Preparación para los planes del proyecto y la calidad. - Revisión de los miembros del personal.

Tabla 23.Tareas de operaciones de SQA

2. Desarrollo y mantenimiento de SQA

Nombre	Descripción
Desarrollo y mantenimiento de SQA	- La decisión de que las normas de SQA se adaptarán, así como el desarrollo y mantenimiento de la organización. - Elaboración de programas para la realización de nuevos procedimientos incluyendo: responsabilidades, actualización, participación de comités y foros. - Seguimiento del desarrollo, cambios en SQA, introducción a nuevos procedimientos y normas de ingeniería del software. - Inicio de actualización, adaptación de procedimientos a los nuevos cambios incluidas la aceptación o supresión de las normas aplicadas por la organización.

Tabla 24.Tareas de desarrollo y mantenimiento

Desarrollo y mantenimiento de SQA se Agrupan en:

- Ingeniera de SQA
- Sistema de información de SQA

Nombre	Tareas
Ingeniería de SQA	Realiza tareas como: - Desarrollo de las soluciones de aplicaciones en herramientas, desarrollo del software, desarrollo para medir la calidad. - Evaluación de la productividad y calidad en el desarrollo de nuevos métodos de mantenimiento y mejoras de métodos. - Realización de métodos para medir la calidad del software y la productividad del equipo. - Apoyo tecnológico a comités durante el análisis de los fracasos y solución a la formulación dada.
Sistema de información de SQA	Realiza tareas como: - Desarrollo de sistemas y procesamiento de datos por la unidad SQA, mantenimiento de internet/intranet. - Nivel de mantenimiento de la unidad SQA - Desarrollo de software y unidades de mantenimiento para albergar datos y procesamientos. Ejemplo: listas, informes, etc. - Facilitar el procesamiento de SQA a la unidad SQA. - Información dada por el desarrollo del software y unidades de mantenimiento. Ejemplo: estimaciones de parámetros y costo de la calidad del software - Actualización de los sistemas de información SQA.

Tabla 25.Tareas de ingeniería de SQA y sistema de información de SQA

SQA administradores

Los administradores SQA son miembros del personal que están interesados en la calidad del software, forma parte de voluntarios que en su tiempo ayudan a promover la calidad, proporcionan el apoyo interno necesario para implementar los componentes de SQA con éxito. Realizan tareas que están relacionadas con la unidad de SQA y la organización.

Nombre	Tareas
Relación con la unidad SQA	- Ayuda a solventar dificultades en la aplicación de los procedimientos y las instrucciones de trabajo de la calidad del software. - Ayuda en la recopilación de datos para el cálculo de métricas SQA. - Informe de los eventos de incumplimiento sustancial y sistemática a la unidad de SQA. - Informe de fallos graves en el software de la calidad a la unidad de SQA.
Relación con la Organización	- Aplicar cambios, actualizaciones de los procedimientos de SQA en la organización e instrucciones de trabajo. - Iniciar mejoras en los procesos de desarrollo, aplicaciones, mantenimiento de soluciones a fallos recurrentes.

Tabla 26.Tareas de personal voluntario interesado en promover la calidad.

SQA los miembros del comité

Nombre	Tareas
Miembros permanentes y comités ad hoc	- Varían considerablemente entre las organizaciones y el tiempo. - Son iniciadas por los distintos órganos, según las circunstancias y necesidades actuales. - Permite comités comunes entre control de cambios de software, procedimientos, herramientas, acciones correctivas, métodos y métricas de calidad. - Actualización de un procedimiento específico de análisis y solución de un fallo en el software, costos de software de calidad y métodos de recopilación de datos.

Tabla 27.Tareas de los miembros de SQA

Miembros del foro SQA

Nombre	Tareas
Miembros del foro SQA	- Mejora los procedimientos y aplicación de SQA. - Métricas de calidad. - Acciones correctivas: análisis de éxito y fracaso. - Calidad de los problemas del sistema, desarrollo en las aplicaciones de nuevas herramientas. - Calidad de los problemas de gestión. - Participación en foros SQA puede ser cerrado. - Participación en foros SQA puede ser abierto en el cual pueden incluir: SQA miembros de la unidad, desarrollo del software y mantenimiento, consultores de ingeniería del software y expertos etc.

Tabla 28.Tareas miembros del foro SQA

4.1.4.3.3 Funciones y responsabilidades

En esta sección se deberá identificar el elemento específico de organización que es responsable de realizar cada tarea.

Establece quien hará qué y quién desempeñará las funciones de trabajo.

Rol	Descripción	Responsabilidad
Jefe de proyecto software	Es el responsable del proyecto, es el mediador entre la dirección y el cliente, asegura que la administración de la calidad se lleve a cabo	Tiene las siguientes funciones: - Participa en la elaboración de la aplicación del plan de desarrollo y el plan de calidad del software además del coste y plazos. - Fases de especificación, diseño e integración. - Define la modalidad de gestión de la configuración - Coordinación de equipos de la gestión. - Fase de validación. - Interlocutor con el cliente.
Responsable de equipo	Desarrollo de las tareas asignada a su equipo	- Tareas de gestión de proyectos es decir planificación y seguimiento entre otros. - Participa en el diseño de la arquitectura. - Fases de diseño, codificación, pruebas en el cual le compete. - Dirección técnica de equipo
Personal de desarrollo	Realiza actividades técnicas dentro del proyecto	
Jefe de la gestión de configuración	Responsable de la coherencia de la configuración	- Identificación de los elementos de configuración. - Administra y controla las actualizaciones. - Controla la biblioteca del proyecto
Jefe de calidad	Garantiza la calidad del software y ante un posible cliente externo	- Dirige el plan de calidad del software con el jefe de proyecto. - Asegura el seguimiento de la calidad como pueden ser: Revisiones, inspecciones, auditoría. - Define las medidas de calidad a realizar. - Interpretación de la calidad en el balance del proyecto.
Administración de SQA	A través del ejecutivo a cargo de la calidad del software	- Define la política de calidad. - Asignación de los recursos. - Asignación de personal. - Seguimiento en la aplicación de políticas de calidad. - Seguimiento de los procedimientos SQA.

		- Soluciona imprevistos, presupuesto, dificultades relacionadas con el cliente.
Ejecutivo a cargo de la calidad del software	Preparación de un programa anual de actividades, presupuesto, preparación de planes de desarrollo, control de la ejecución del programa.	- Establecer los objetivos del sistema de SQA. - Determinar si el programa de actividades es adecuado a las características y el alcance de los servicios de subcontratistas y las compras de software previstas. - Aprobar la versión definitiva del programa anual de actividades y el presupuesto SQA. - Desarrollo de métricas de calidad de software adecuadas para la evaluación de las nuevas herramientas y normas. - Revisión de los avances de los proyectos de adaptación SQA. - Determinar la mano de obra y otros recursos previstos para la aplicación del programa de SQA. - Revisión de las tendencias que se espera que afecten a la calidad de la organización de software en un futuro próximo. - Elaboración de nuevos procedimientos adecuados a las nuevas herramientas y estándares de SQA.

Tabla 29.Funciones y responsabilidades

4.1.4.3.4 Aseguramiento de la calidad estimación de los recursos

En esta sección se deberá proporcionar la estimación de los recursos y los costes de la garantía de calidad y tareas de control de calidad.

La estimación de costo más sencilla es la que proporciona un coste fijo desde el inicio, sin permitir desviaciones en ningún ámbito. Aunque las organizaciones muy competentes tienen suficientes aptitudes para cambiar las variables restantes como son: capacidad, programas de tiempo y calidad; para cumplir con un coste predeterminado.

El proceso de estimar los costos esto es, para capacidades, control de calidad y programación a menudo comienza desde la concepción del proyecto y continúa aun después de iniciada la codificación. Cuanto más se sepa de los requisitos y más diseño se realice, más preciso será el costo.

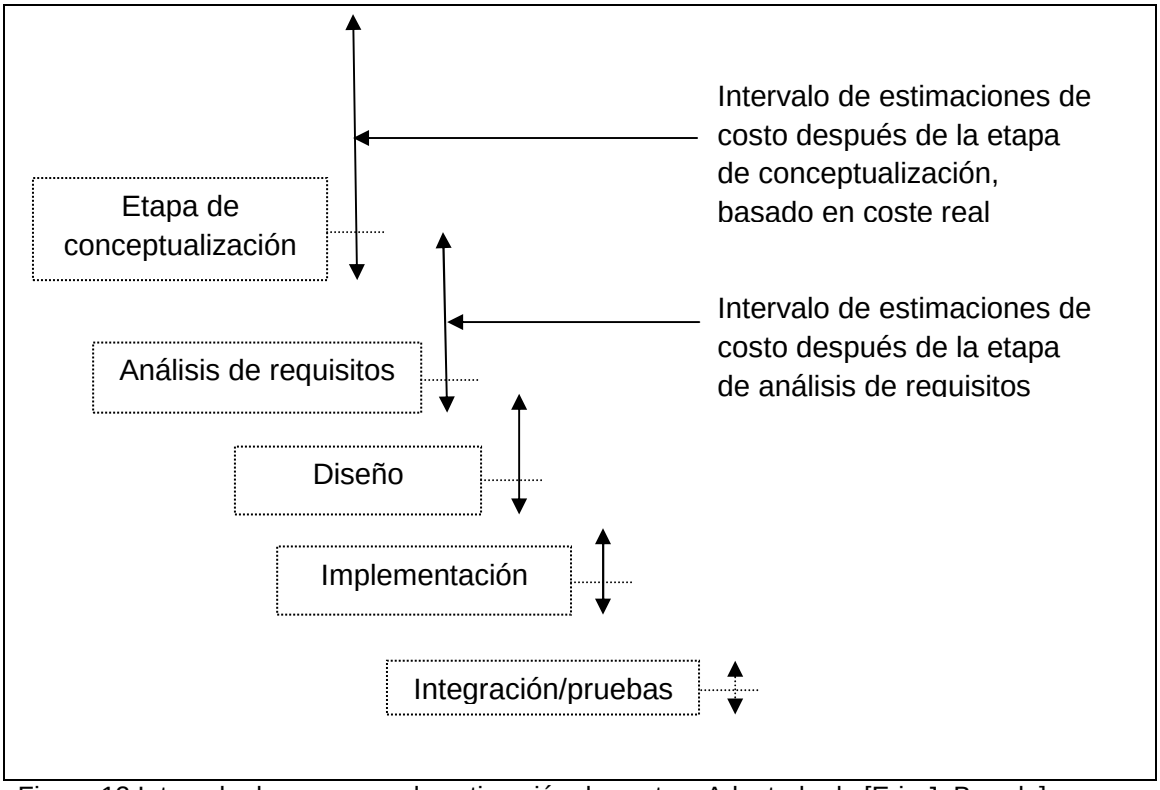


Figura 12.Intervalo de errores en la estimación de costes. Adaptado de [Eric J. Braude]

El error de la estimación se debe a un estudio realizado por *Boehm, Barry, Ingeniería del software de economía, NJ: Prentice Hall, 1981.* Para perfeccionar la estimación del coste de un proyecto se usarán varias técnicas, lo que significa disminuir la altura de las líneas verticales en la figura 12. Sólo hasta el final del desarrollo se puede tener una confianza completa en las estimaciones.

Una buena manera de enfocar la estimación de costos del proyecto durante las primeras etapas es desarrollar estimaciones de varias maneras independientes y después combinar los resultados. Incluso se pueden ponderar las estimaciones obtenidas de acuerdo con el nivel de confianza personal en cada una de ellas.

En la prevención y evaluación disminuyen los costes de fallos

Los costes de la calidad son definidos en tres categorías:

- Coste de la conformidad
- Coste de la no conformidad
- Coste de la oportunidad desperdiciada

1. Costes directos de la calidad: La dirección tiene control directo para garantizar que los productos y servicios aceptables se remitan al mismo. Son los que mejor se comprenden y se utilizan.

    - Costes de control de la calidad: Son los costes de prevención y evaluación.

- Costes resultantes de la no calidad: Costes a los que hace frente una empresa, por no realizar las cosas bien desde el principio; se clasifican en :

    - Costes de fallos internos: Se refiere como consecuencia a los errores detectados antes de que sea aceptado por el cliente, como pueden ser: costes por suministro de materiales, pérdidas evitables en el proceso, baja de precios, repetición del trabajo, etc.
    - Costes de fallos externos: Una vez entregado el producto o servicio al cliente es inaceptable, debido a: quejas tanto dentro como fuera de la garantía, mantenimiento del producto o servicio, retirada del producto, etc.

2. Costes indirectos de la calidad: forman parte del coste de calidad del ciclo de vida del producto o servicio. Se basan en tres aspectos importantes: coste en el que está presente el cliente, coste de la insatisfacción del cliente, coste de la pérdida de imagen.

Costes de la calidad del software

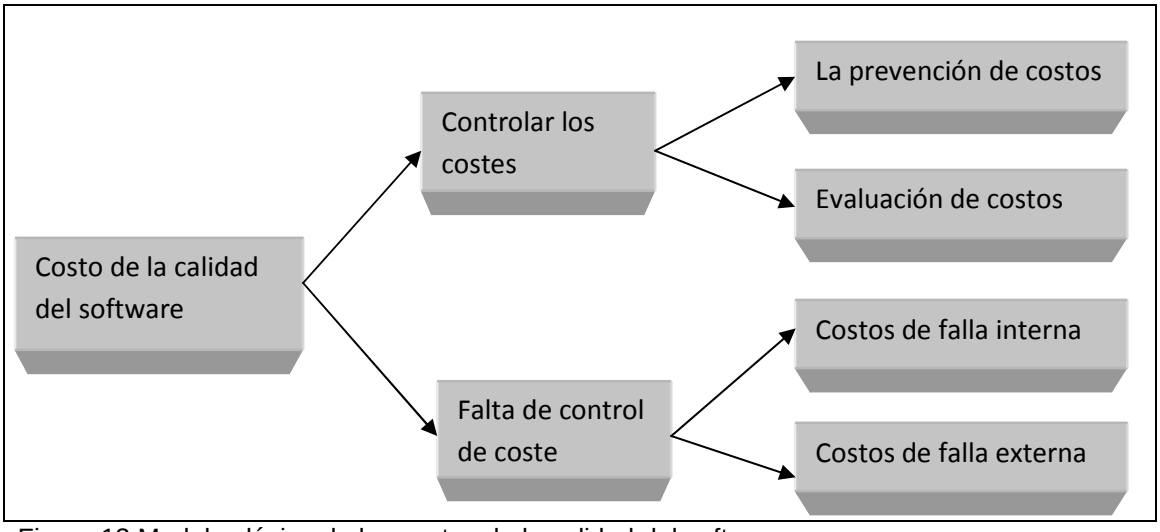


Figura 13.Modelo clásico de los costes de la calidad del software

– **La prevención de costos**

Los costos de prevención incluyen una mejora de la infraestructura de la calidad del software, así como la ejecución de las actividades para su funcionamiento. Una parte importante de las actividades realizadas por el equipo de SQA es de carácter preventivo. Los costes de prevención incluyen:

1. Inversión en el desarrollo de componentes en la infraestructura de SQA, y actualización periódica de los componentes:

    - Métricas del software de la calidad.
    - Aseguramiento de configuración del software del sistema.

- Dispositivos de apoyo.
- Procedimientos e instrucciones de trabajo.

2. Actividades de SQA preventivas de la aplicación incluyen:

   - Instrucciones a los miembros del equipo en temas de SQA.
   - Asesoramiento sobre las cuestiones de SQA que proporcionan los jefes de equipo y otros miembros de la organización.

3. Control del sistema SQA través del desempeño de:

   - Revisiones internas de calidad.
   - Auditorías externas de calidad por los clientes y las organizaciones de SQA del sistema.
   - Gestión de control de calidad.

– **Evaluación de costos**

Los costos de evaluación se dedican a la detección de errores de software en proyectos concretos o sistemas de software. Los costes más frecuentes en la evaluación comprenden:

1. Comentarios

   - Revisiones del diseño formal.
   - Comentarios de personal experto.
   - Revisiones, inspecciones etc.

2. Costos de pruebas de software

   - Integración de pruebas.
   - Unidad de pruebas.
   - Software de pruebas de sistema.
   - Pruebas de aceptación es decir pruebas que desarrolla el cliente.

3. Costo SQA de los participantes externos, fundamentalmente revisiones de diseño y pruebas de software, estas actividades son realizadas por:

   - Subcontratista.
   - Proveedores de sistema de software.
   - El cliente como un integrante más en el desempeño del proyecto.

– **Costos de fallos internos**

Los costes internos son fracasos ocasionados al corregir los errores que han sido detectados por las revisiones de diseño, pruebas de

software y pruebas de aceptación realizadas antes de que el software haya sido instalado en las instalaciones del cliente. Los costos típicos de fallos internos son:

- Los costos de las correcciones del nuevo diseño o siguientes al diseño de resultados de la revisión y la prueba.
- Los costos de la nueva programación o la corrección de los programas en respuesta a la obtención de los resultados de las pruebas.
- Los costos de la revisión del diseño repetido y nuevas pruebas (pruebas de regresión). Las revisiones de diseño repetido o pruebas de software son consecuencia directa de un mal diseño, la calidad del código interno se considera costos internos de fracaso.

– **Los costes externos de fracaso**

Los costes externos de fracaso suponen los costos de corregir deficiencias detectadas por los clientes o equipos de mantenimiento después de que el sistema de software se ha instalado en las instalaciones del cliente. Estos costos pueden ser clasificados en abiertos (o de manifiesto) y ocultos.

En la mayoría de los casos, la medida de los costos ocultos es muy superior que la de los costos de manifiesto. Esta diferencia se debe, entre otras cosas, a la dificultad de estimar costos ocultos de fallos externos en comparación con el fracaso de los costes externos, que son fácilmente registrados o estimados.

La estimación de costes externos ocultos es un fracaso que pocas veces viene como consecuencia de ello. Por consiguiente los costos de fallos externos tienden a cubrir los siguientes aspectos:

- Resolución de las reclamaciones de los clientes durante el periodo de garantía; esto implica una revisión de queja, muchas veces estas quejas son realizadas por directrices que se encuentran en el manual de instrucciones.
- Corrección de errores de software detectados durante el funcionamiento normal. La corrección de la participación de código (incluidas las pruebas del software de corrección), seguida por la instalación del código de corrección o el reemplazo de la versión equivocada de la versión correcta se realizan a menudo en la instalación del cliente.
- Corrección de errores de software después de que el período de garantía ha terminado, aunque la corrección no está cubierta por la garantía.

Aplicación de los costos del sistema de calidad del software

La aplicación de un nuevo costo del sistema de software de calidad implica:

- Asignación de la responsabilidad de informar y recoger datos de calidad de costos.
- Seguimiento
  - Apoyo para solventar problemas de implementación y ofrecer información cuando sea necesario.
  - Revisión de los informes de costes, clasificación y registro.

- Actualización y revisión de las definiciones de los elementos de coste.

Dependiendo de las medidas tomadas en respuesta a las conclusiones del modelo

Los resultados obtenidos tras el análisis de los informes de calidad del software basado en comparaciones con períodos anteriores. Tienen su origen en la aplicación del costo de la calidad del software. Un incremento de los costos de control se espera que produzca una disminución en fracaso de los costes de control y viceversa: una disminución en los costos de control se espera que conduzca a un aumento en fracaso de los costes de control.

Por otra parte, el efecto de modificaciones en los costos de control se espera que varíen según el nivel de calidad del software que se espera. Esta relación se espera obtener un mínimo costo total de la calidad del software, un coste puede lograr un nivel de calidad especificado, el nivel de software de calidad óptima. La Figura 14 ilustra la gráfica del costo del concepto de equilibrio de calidad del software y las relaciones entre el control y el fracaso de los costes de control de todos los niveles de calidad.

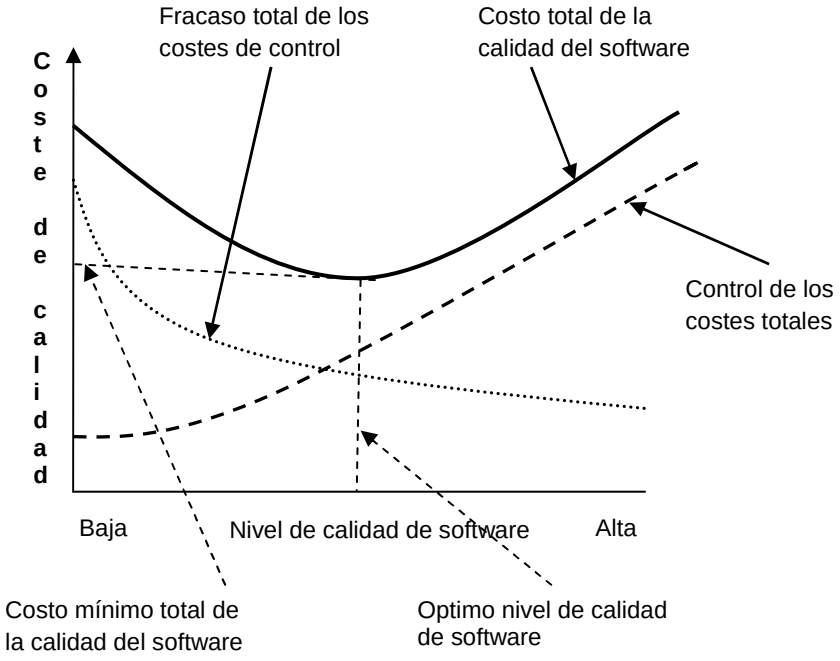


Figura 14. Variación de los costes controlables de la calidad.

Problemas en la aplicación del costo de las métricas de calidad de software

La aplicación de un modelo de costo de la calidad del software está acompañada de problemas que superar, sea cual sea el sector. Estos afectan a la precisión e integridad de los datos de costes de calidad causada por:

- la inexactitud o la identificación incompleta y clasificación de los costes de calidad.
- Negligencia en la presentación de informes por los miembros del equipo y otros.
- Sesgos en los informes del costo del software especialmente censurando los costes tanto internos como externos.

Causa de la desviación de calendario	Clase de los costes de calidad
Los cambios introducidos en las especificaciones del proyecto durante el desarrollo	No hay costos de fallo interno; la responsabilidad del cliente de costes de fallos
Instalación del cliente con retraso de comunicación y otros equipos, y retrasos en la contratación y formación del personal.	No hay costes de fallos internos, la responsabilidad del cliente de costes de fallos
Mal desempeño por el equipo de desarrollo, que requieren reanudación amplio y correcciones de software.	Costes externos fracaso
Propuesta de proyecto basada en programas y presupuestos poco realistas	Fracaso costes de gestión
Contratación o falta de personal suficiente o la confianza en profesionales de la empresa cuya versión de otros proyectos no satisface las necesidades del proyecto	Fracaso costes de gestión

Tabla 30.Las causas típicas de los retrasos y costes asociados

4.1.4.4 Documentación

Especificar de manera clara y objetiva toda la información que se generará y que se usará en cada fase del proyecto, se debe desglosar por fases y se generará mediante los siguientes documentos:

- *PAQS*
- *PACS*
- *PAPS*
- *ERS*
- *DDS*
- *DPS*
- *Manual de Usuario*
- *Plan de mantenimiento*
- *PVVS (Plan de Verificación y Validación) será generada y mantenida por una organización independiente de la organización de SQA.*

4.1.4.4.1 Propósito

Define y lista los documentos más importantes dentro del SQAP, que deben ser creados para garantizar la calidad del proyecto.

El propósito es definir la documentación que se utilizará para asegurar la calidad.

Nivel	Lista de documentos
Documentos de anteproyecto	- Informe de revisión de contrato. - Contrato de desarrollo del software. - Contrato de mantenimiento del software. - Contrato de subcontratación del desarrollo del software. - Plan de desarrollo del software.
Documentos de proyecto del ciclo de vida	- Documentación de requisitos del sistema. - Documentos de los requisitos del software. - Documento de diseño arquitectónico. - Documento de diseño detallado. - Descripción de la base de datos. - Plan de pruebas del software. - Diseño de revisión de informe. - Seguimiento de los registros de elementos de diseño de acción de revisión. - Procedimiento de software de prueba. - Informe de software de prueba. - Manuales de usuario del software. - Manuales de mantenimiento de software. - Plan de instalación de software. - Documento de descripción de la versión. - Solicitudes de cambio de software. - Solicitudes de mantenimiento de software. - Servicios de informes de mantenimiento. - Registros de las evaluaciones de los subcontratistas.
Documentos de la infraestructura SQA	- Procedimientos del SQA. - Plantilla de la biblioteca. - Formularios de biblioteca de SQA.
Software de documentos de gestión de calidad	- Informes de progreso. - Informes de métricas de software.
Documentos sistema de auditoría del SQA	- Informe de revisión de la gestión. - La calidad del informe de auditoría interna. - Certificación externa del informe de auditoría de SQA.
Documentos de los clientes	- Documentos de los proyectos del software de oferta. - Cambio del software por petición del cliente.

Tabla 31.Tipos de documentos SQA

4.1.4.4.2 Requisitos mínimos de documentación

Para garantizar que la aplicación del software satisface los requisitos técnicos, la siguiente documentación es requerida como mínimo.

Se lista toda la documentación del proyecto, ya que es la que asegurará la calidad del producto.

4.1.4.4.2.1Descripción de los requisitos del software (SRD)

El SRD debe especificar los requisitos para un determinado producto de software, proyecto o conjunto de proyectos que realizan ciertas funciones en un entorno específico.

La definición de requisitos es una especificación técnica de los requisitos que debe cumplir el producto, su principal objetivo es definir los requisitos de manera específica y consistente ya que los requisitos del producto deben desarrollarse de una manera concisa y sin ambigüedades.

Por tanto la especificación de requisitos se basa en la definición del sistema y establecerá "qué" es el producto sin implicar "como" es este.

El documento de especificación de requisitos es el encargado de recoger todo el fruto de trabajo realizado durante la etapa de análisis del sistema de una forma integral en un único documento.

Por tanto el SRD es un documento fundamental y es el punto de partida del desarrollo de cualquier sistema software, debe cubrir todos los objetivos en el análisis de requisitos del software preferiblemente se recomienda el uso del estándar IEEE 830-1998 ya analizado.

El SRD deberá ser revisado con frecuencia durante la aplicación, por lo tanto es conveniente que se redacte de una forma fácil que se pueda modificar, también debe facilitar la labor de la verificación del cumplimiento de las especificaciones. Todo ello hace que la mejor manera de redactar el SRD sea en forma de contrato con sus distintas clausulas organizadas y según el carácter de los requisitos.

Un documento de requisitos debe ser:

- Correcto y completo: Un conjunto de requisitos incorrecto e incompleto puede producir un producto del proyecto que satisfaga los requisitos pero no los del cliente.
- Consistente: Una especificación inconsistente establece requisitos contradictorios en diferentes partes del documento.
- No ambiguo: Requisitos ambiguos se prestan a diferentes interpretaciones por diferentes personas.
- Funcional: Debe describir lo que se necesita sin implicar como se cumplirá el sistema con los requisitos.

- Verificable: Comprobar que los requisitos cumplen las necesidades del cliente y verificar si los productos cumplen los requisitos.
- Rastreable: Cada requisito debe ser rastreable tanto los requisitos del cliente como la definición del sistema.
- Fácilmente modificable: Que se puedan cambiar los requisitos sin alterar la relación con otros.

Heninger (1980) sugiere seis requisitos que debe satisfacer un documento de requisitos de software:

1. *Especificará únicamente el comportamiento externo del sistema.*
2. *Especificará las restricciones de la implementación.*
3. *Será fácil de cambiar.*
4. *Servirá como herramienta de referencia para los mantenedores del sistema.*
5. *Registrará las previsiones del ciclo de vida del sistema.*
6. *Caracterizará las respuestas aceptables para los eventos no deseados.*

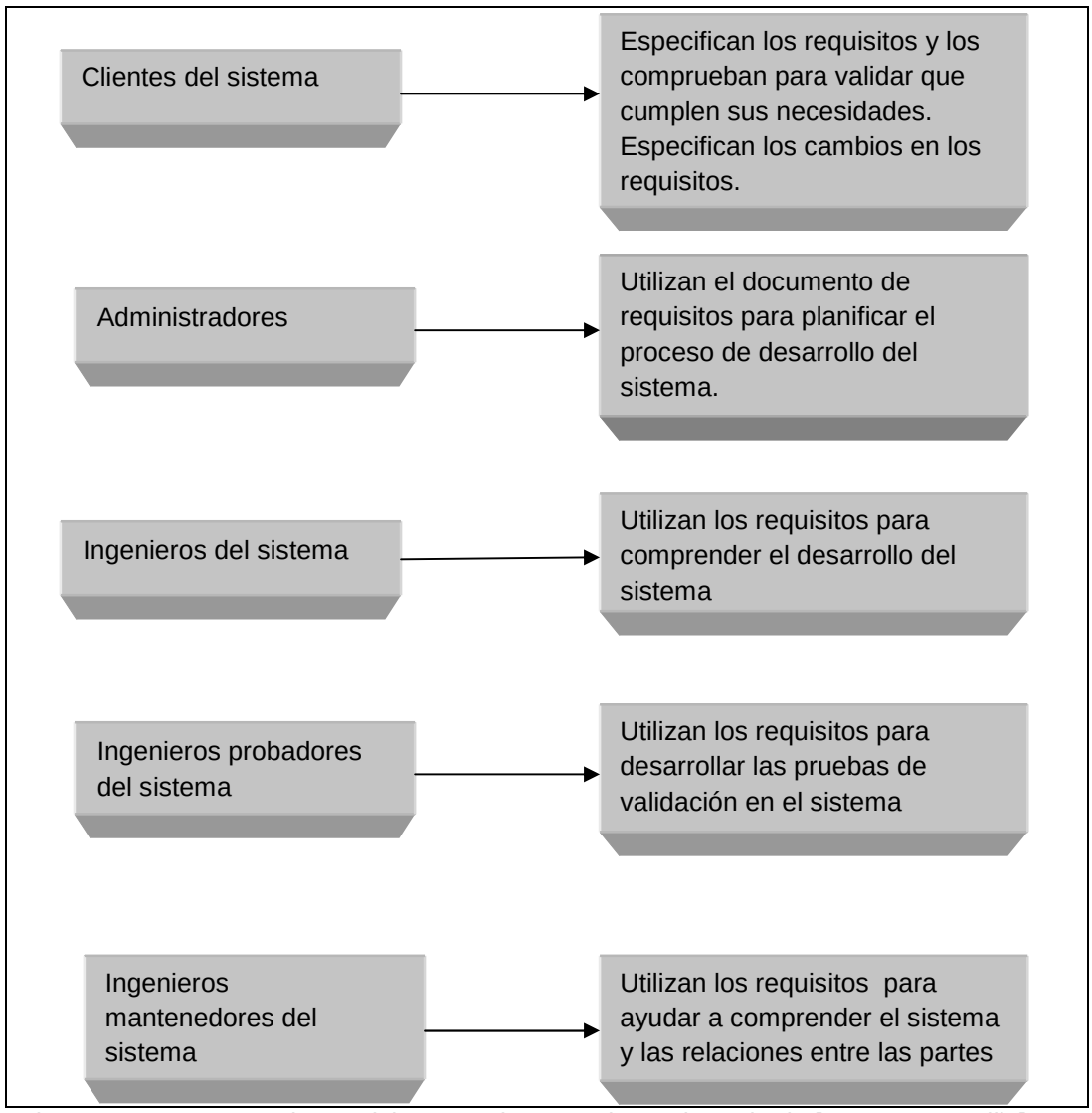


Figura 15.Documento de requisitos para los usuarios. Adaptado de [Ian Sommerville]

4.1.4.4.2.2Descripción de diseño de software (SDD)

El SDD debe describir cómo el software será capaz de satisfacer los requisitos de la SRD. El SDD debe describir los componentes y subcomponentes del diseño de software, incluyendo bases de datos internas e interfaces

Es un documento de diseño que tiene como resultado fundamental en la etapa de diseño ser utilizada en sucesivas etapas del proyecto, cuando la complejidad del documento de diseño del software resulte muy extenso se describirá la estructura del sistema de una manera jerarquizada. Se presenta una serie de documentos y las empresas eligen el organismo de estandarización externa:

- Departamento de Defensa Norteamericano.
- Agencia Espacial Europea. Las normas ESA establecen el empleo de un documento de diseño arquitectónico (ADD) para describir el conjunto del sistema y otro documento de diseño detallado (DDD).
- O internas creadas por la propia organización para uso corporativo o para un tipo específico de proyectos.

Uno de los riegos que se pueden correr es que la documentación sea demasiado extensa, para solventarlo y evitar correr riegos en cada diseño se deben realizar guías y convenios de documentación. El proceso de diseño del software consiste en realizar un buen diseño aplicando principios fundamentales de diseño, metodología y revisión cuidadosa.

En la documentación de diseño tiene que quedar plasmada la prevención de defectos antes de que ocurran, poniendo atención en la necesidad del que el personal encargado del diseño de los procesos y productos los estandarice para conseguir que sean fiables.

En el diseño se establecen dos tipos de documentación:

- Documento de diseño arquitectónico (ADD):

  Los requisitos proporcionan el marco de trabajo para el diseño arquitectónico.

  En el diseño arquitectónico el objetivo es especificar una estructura del sistema que satisfaga los requisitos, las especificaciones del diseño externo es el paso de transición de la definición de requisitos (el “que” detallado) a la arquitectura del sistema y las restricciones de la instrumentación que es escribir el código fuente y la documentación interna de modo que la concordancia con sus especificaciones sea fácil de verificar y que se facilite la depuración, pruebas y modificaciones.

Se presenta el siguiente índice:

1. INTRODUCCIÓN
   1.1 Objetivo
   1.2 Ámbito
   1.3 Definiciones, siglas y abreviaturas
2. PANORÁMICA DEL SISTEMA
3. CONTEXTO DEL SISTEMA
   …..
   3.n Definición de interfaz externa

4. DISEÑO DEL SISTEMA
   4.1 Metodología de diseño de alto nivel
   4.2 Descomposición del sistema
5. DISEÑO DE LOS COMPONENTES
   …..
   5.n Identificador del componente
      5.n.1 Tipo
      5.n.2 Objetivo
      5.n.3 Función
      5.n.4 Subordinados
      5.n.5 Dependencias
      5.n.6 Interfaces
      5.n.7 Recursos
      5.n.8 Referencias
      5.n.9 Procesos
      5.n.10 Datos
6. VIABILIDAD Y RECURSOS ESTIMADOS
7. MATRIZ REQUISITOS/COMPONENTES

Tabla 32.Documento diseño Arquitectónico (ADD)

- Documento de diseño detallado (DDD)

  El diseño detallado se desarrolla a partir del diseño arquitectónico. El diseño detallado proporciona los detalles algorítmicos, la representación de datos y el empaquetamiento del producto de la programación.

  Se presenta el siguiente índice:

Parte 1. DESCRIPCIÓN GENERAL
1. INTRODUCCIÓN
   1.1 Objetivo
   1.2 Ámbito
   1.3 Definiciones, siglas y abreviaturas
   1.4 Referencias
   1.5 Panorámica

2. NORMAS, CONVENIOS Y PROCEDIMIENTOS
   2.1 Normas de diseño de bajo nivel
   2.2 Normas y convenios de documentación
   2.3 Convenios de nombres
   (Ficheros, programas, módulos, etc.)
   2.4 Normas de programación
   2.5 Herramientas de desarrollo de software

Parte 2. ESPECIFICACIONES DE DISEÑO DETALLADO
n. Identificador del componente
   n.1 Identificador
   n.2 Tipo
   n.3 Objetivo
   n.4 Función
   n.5 Subordinados
   n.6 Dependencias
   n.7 Interfaces
   n.8 Recursos
   n.9 referencias
   n.10 Proceso
   n.11 Datos

Tabla 33.Documento de diseño detallado (DDD)

En el documento de diseño detallado cabe destacar la importancia de las normas, convenios y procedimientos que se aplicaran durante el progreso del sistema. La importancia del trabajo en equipo es fundamental para que se tenga una estructura coherente y homogénea.

Esta actividad de normas convenios y procedimientos debe ser la primera que se realice en el diseño detallado antes de comenzar el diseño propiamente dicho.

4.1.4.4.2.3 Planes verificación y validación (V & V)

Los procesos de verificación y validación se utilizan para determinar si los productos de software desarrollados se ajustan a sus necesidades, y si los productos software cumplen con el uso previsto y las expectativas del usuario.

El plan de verificación y validación es un documento para determinar que los requisitos del cliente han sido implementados correctamente.

Por lo cual los objetivos de la verificación y validación son:

- Detección y corrección de los defectos lo antes posible en el ciclo de desarrollo del software.
- Reducir los riesgos.
- Mejorar la calidad y fiabilidad del software.
- Valorar los cambios propuestos y sus consecuencias.

En el plan de verificación y validación se describirán los procedimientos a utilizar:

- La verificación:
    - Según los requisitos descritos en el documento, el software está de acuerdo con su especificación.
    - Satisfacción de los requisitos funcionales y no funcionales.
    - Garantiza la consistencia entre los productos desarrollados y sus requisitos especificados en la fase de especificación.
    - Técnica más utilizada: Revisión del software.

- La validación:
    - Los requisitos y pruebas durante el ciclo de desarrollo descrito en el documento satisfacen las expectativas del cliente.
    - Garantiza que el software final satisfaga los requisitos del sistema.
    - Técnica más utilizada: Pruebas del software.

La documentación de verificación y validación tiene que quedar plasmada:

- Especificación del diseño de pruebas.
- Especificación del caso de prueba.
- Especificación de los procedimientos de prueba.
- Plan de pruebas.
- Documento de anomalía.
- Documento final de pruebas.

4.1.4.4.2.4 Informe de verificación de resultados e informe de validación de resultados

El informe de verificación de los resultados debe describir los resultados de las actividades de verificación del software realizado de acuerdo con el plan de verificación. El informe de validación de los resultados debe describir los resultados de la prueba de software o de validación llevada a cabo de acuerdo con el plan de validación.

- Plan de verificación del software

1. Requisitos que se verificarán. 2. Plan de verificación de diseño. 3. Plan de pruebas de código fuente. 4. Criterio para determinar las pruebas. 5. Verificación de la documentación aportada. 6. Herramientas, técnicas y métodos que se emplearán.

Tabla 34.Informe plan de verificación

Establece los métodos que se utilizarán y la obtención de los resultados esperados en la verificación de cada requisito que se haya establecido en la especificación para la producción del producto software.

- Plan de validación de los resultados de la prueba del software

1. Requisitos que se verificarán. 2. Casos de prueba para cada uno de los requisitos. 3. Obtención de los resultados esperados en cada caso de prueba 4. Habilidades demostradas por cada prueba

Tabla 35.Informe plan de validación

Confirma el cumplimiento de cada uno de los requisitos y aceptación de cada una de las pruebas realizadas durante el ciclo de desarrollo y que satisfaga las expectativas del cliente.

4.1.4.4.2.5 Documentación del usuario

La documentación de usuario debe describir el control de entradas de datos, secuencias de entrada, opciones, las limitaciones del programa, y cualquier otra información esencial para el producto de software.

La documentación de usuario debe especificar y describir los datos y entradas de control requeridos, así como la secuencia de entradas, opciones, limitaciones de programa y otros elementos necesarios para la ejecución exitosa del software.

Todos los errores deben ser identificados y las acciones correctivas descritas.
Como resultado del proyecto, el cliente obtendrá una documentación de acuerdo a los requisitos específicos del proyecto.

La documentación de usuario, proporciona un mecanismo de control entre el usuario final y equipo de desarrollo, para lo cual se utiliza información del sistema, resultados de las distintas etapas del ciclo de vida como son prototipos, pruebas, informes, despliegues entre otros.

La documentación de usuario forma parte de las especificaciones de requisitos.

A continuación se presenta el esquema del documento de usuario.

- Parte 1:

- Introducción.
- Panorama y exposición del producto.
- Terminología y características básicas.
- Resumen de informes y despliegues.
- Manual de usuario.

- Parte 2:
- Pasos iniciales.
- Arranque.
- Modo de ayuda.

- Parte 3:

- Comandos.
- Diálogos.
- Informes.

- Parte 4:
- Características especiales.

- Parte 5:
- Sintaxis de los comandos y opciones de sistema.

Tabla 36.Documento manual de usuario

4.1.4.4.2.6Plan de gestión de configuración del software (SCMP)

El SCMP debe documentar, las actividades y formas que deben llevarse a cabo, quién es el responsable de distintas tareas, calendario de eventos, y qué recursos se utilizarán.

1. Introducción.
2. Administración de la gestión de configuración del software.
   - 2.1 Organización.
   - 2.2 Responsabilidades de la gestión de configuración del software.
   - 2.3 Políticas aplicables, directivas y procedimientos.
3. Actividades de la gestión de configuración del software.
   - 3.1 Identificación de la configuración.
   - 3.2 Control de la configuración.
   - 3.3 Responsabilidades del estado de la configuración.
   - 3.4 Auditorías y revisiones de la configuración.
   - 3.5 Control de interfaz.
   - 3.6 Control de contratista/ proveedor.
4. Programa de tiempos de la gestión de configuración del software.
5. Recursos de la gestión de configuración del software.
6. Mantenimiento del plan de la gestión de configuración del software.

Figura 16.Tabla de contenido del plan de gestión de configuración del software IEEE 828-2005

1. Gestión del proceso de gestión de configuración del software

La gestión de configuración del software es uno de los procesos más importantes para toda organización, es un elemento indispensable de garantía de calidad, es responsable de controlar los cambios para su desarrollo e integridad de un producto identificando algunos de sus elementos entre ellos cabe destacar:

- Especificación del sistema.
- Planificación del proyecto.
- Especificación de requisitos.
- Manual de usuario (preliminar).
- Especificación de diseños.
- Descripción de la base de datos.
- Pruebas.
- Mantenimientos.
- Estándares y procedimientos.
- Etc.

Tareas de gestión de configuración del software

1.1 Contexto de organización para SCM

La dirección debe coordinar la elaboración de un plan para la implantación del sistema de calidad, en el cual cabe destacar: Un plan de acciones concretas, establecimientos de cronogramas de actividades y previsión de recursos humanos y financieros que serán necesarios. Se necesita comprender el contexto de la organización y la relación de los elementos de la organización.

Los procesos de soporte en la ingeniería del software se pueden clasificar de distintas maneras, aunque la responsabilidad de realizar algunas tareas de la gestión de configuración del software se podría asignar a otras tareas de la organización haciendo hincapié en la organización de desarrollo en el cual es un elemento definido en la organización.

Se desarrolla el software como parte de un sistema mayor que contiene hardware y firmware en este caso las actividades de gestión de configuración del software se realizan en paralelo con la gestión de configuración de las actividades del hardware y firmware y además debe ser consistente con el sistema.

1.2 Restricciones y consejos para el proceso de la SCM

Los consejos y restricciones pueden venir de distintas fuentes, los procedimientos y los estándares a nivel de organización pueden tener influencia en el diseño e implantación de los procesos en un proyecto

determinado. El contrato entre el cliente y el proveedor podrían contener pautas que afecten los proceso.

1.3 Planificar la SCM

La planificación de un proceso para un proyecto debe ser consistente con las restricciones y consejos.

Las actividades deben estar cubiertas en esta tarea de planificar la SCM: Establecimiento de un calendario de actividades, recursos, selección de herramientas, implementaciones, control de proveedores, etc.

Los resultados obtenidos en la planificación de las actividades se introducen en un plan de la gestión de configuración el cual está en todo momento revisado y auditado. El plan de SCM deberá describir los métodos para nombrar los ítems o elementos con el propósito de almacenamiento, seguimiento, recuperación, reproducción y distribución. Este plan precisa cuestiones tales como:

- Objetivos que se persiguen con la implantación del sistema.
- Determinadas fases de que debe constar.
- Responsables de documentar el sistema.
- Responsabilidad para la puesta en marcha.
- Recursos.
- Proceso de auditoría y certificación.
- Y cualquier otra eventualidad que se considere importante.

1.3.1 Organización y responsabilidades de la SCM

La organización de SCM:

Se describe el contexto de la organización, tanto técnicas como de gestión, en el que las actividades previstas de SMC se llevarán a cabo. El Plan deberá identificar los siguientes aspectos:

- Todas las unidades organizativas que participan en él o son responsables de cualquier actividad de SMC en el proyecto.
- El papel funcional de estas unidades organizativas dentro de la estructura del proyecto.
- Las relaciones entre las unidades de organización y las interfaces de la aplicación en las relaciones.

Las unidades de la organización pueden consistir en un vendedor y el cliente, el contratista principal y subcontratistas, o diferentes grupos dentro de una organización.

Los organigramas de la organización, complementada por las declaraciones de la función, las funciones y relaciones, puede ser una manera eficaz de presentar esta información. Si el plan para mantener el diagrama

organizativo muestra una cierta organización o grupo de gestión asumiendo esta responsabilidad, será adecuado referenciar en el plan en vez de ponerlo en el diagrama actual en el documento, en el cual debe mantenerse cada vez que otro grupo de diagramas sea actualizado.

Responsabilidades de la SCM

Especificación de la asignación de las actividades de la gestión de configuración del software en las unidades organizativas. Cada actividad debe facilitar el nombre del trabajo que desempeña en la actividad.

Una matriz que conecte las organizaciones sobre las funciones, actividades y tareas en SCM esto es de gran importancia para documentar las responsabilidades de SCM.

Del comité de revisión o de la organización especial establecida para la realización de actividades de SCM en este proyecto, el plan deberá describir lo siguiente:

- Propósito y los objetivos.
- Miembros y afiliaciones.
- Período de efectividad.
- Alcance de la autoridad.
- Los procedimientos operativos.

1.3.2 Recursos y planificación de la SCM

- **Recursos del SCM**

En la gestión de configuración del software se obtiene información sobre la identificación de los recursos, infraestructura, las herramientas software, técnicas, equipos de personal y capacitación necesario para la aplicación de las actividades específicas de SMC.

SMC puede llevarse a cabo, mediante un entorno general o infraestructura, por una combinación de herramientas de software y manual de procedimientos. Las herramientas pueden ser SMC específicas o integradas, en ayudas generales del proyecto; los cuales pueden ser recursos estándares o recursos de la organización especialmente adquiridos o construidos para este proyecto.

Las herramientas se pueden aplicar al control de acceso, desarrollo de la documentación y seguimiento, el control de código, la generación de línea de base del sistema, el cambio de procesamiento, la comunicación y la autorización, el cambio/el problema de seguimiento e informes de estado, archivo, retención y recuperación de los elementos controlados, o la SMC proceso de planificación en sí mismo.

La infraestructura de SMC debe ser planificada y documentada, considerando factores como: funcionalidad, rendimiento, seguridad, restricciones de seguridad, disponibilidad, necesidades de espacio, equipo, costos y tiempo. La infraestructura será el mantenimiento, seguimiento y las modificaciones necesarias para asegurar que sigue cumpliendo con los requisitos del proceso de SCM.

Para cada tipo de actividad SCM identificada, el plan deberá especificar qué herramientas, técnicas, equipos de personal y capacitación se requieren y cómo cada uno de los recursos serán proporcionados u obtenidos.

Para cada herramienta de software, desarrollada dentro del proyecto o fuera del proyecto, el plan deberá describir o hacer referencia a sus funciones y deberá identificar los controles de configuración para ser colocados en la herramienta.

- **Mantenimiento de la Planificación de SCM**

El mantenimiento de la Planificación de SCM identifica las actividades y responsabilidades necesarias para asegurar la continuidad de la planificación de SMC durante el ciclo de vida del proyecto. El Plan incluirá cambios realizados en el plan y establecerá lo siguiente:

- ¿Quién es el responsable de la supervisión del plan?
- ¿Con qué frecuencia se desarrollan las actualizaciones?
- ¿Cómo los cambios en el plan han de ser evaluados y aprobados?
- ¿Cómo los cambios en el plan han de ser realizados y comunicados?

El Plan debe ser revisado al comienzo de cada fase del proyecto de software, se cambiará en consecuencia, y es aprobado y distribuido al equipo del proyecto. Debido a la importancia de un plan de SCM, todos los cambios en este documento deben aprobarse por todo el equipo de la gestión de configuración.

En vista a la organización de desarrollo de software que se desea alcanzar en el nivel 5 de CMM, el líder de la configuración realizará lo siguiente para mejorar la gestión de configuración.

- Revisar la efectividad del plan.
- Cuantificar los defectos de pérdidas del plan.
- Enumerar los beneficios de las mejoras.
- Proporcionar beneficios de costos respectivamente a su efecto en las mejoras.
- Asignar prioridades a razón del costo/beneficio de todos los cambios que se hayan sugerido.
- Investigar herramientas de gestión de configuración.

1.3.3. Selección e implementación de herramientas

Las herramientas se pueden combinar con herramientas manuales, herramientas automáticas, herramientas integradas que cubran las necesidades de los que participen en el proceso. Estas herramientas son de gran importancia y a la vez difíciles de establecer con respecto al tamaño del proyecto ya que empiezan a crecer durante su desarrollo y el entorno se hace muy complejo. Estas herramientas proporcionan ayuda en:

- La biblioteca de la SCM.
- Procedimientos de aprobación y de petición de cambios del software.
- Tareas de gestión de cambio y código.
- Informes del estado de configuración del software y reunión de medidas de la SCM.
- Auditoría de la configuración del software.
- Gestión y seguimiento de la documentación del software.
- Construcción del software.
- Gestión y seguimiento de los lanzamientos del software y su distribución.

Estas herramientas pueden proporcionar medidas en la mejora de los procesos *Royce propone siete medidas centrales útiles para gestionar los procesos.*

La información disponible de la diversidad de herramientas de la SCM es afín al indicador de Trabajo y Progreso de Royce y a sus indicadores de calidad de Cambio de Tráfico y Estabilidad, Ruptura y Modularidad, Repetición del Trabajo y Adaptabilidad y TMEF (Tiempo medio entre fallos) y Madurez. Los informes para estos indicadores se pueden organizar de varias maneras, como elemento de configuración del software o por tipo de cambio requerido.

1.3.4 Control Proveedores/Subcontratas

Tanto para el producto subcontratado como adquirido, el plan de gestión de configuración del software definirá las actividades para agregar los elementos externos en elementos de la configuración del proyecto para acoplar los cambios de esos elementos en sus organizaciones de desarrollo.

El plan describirá para el software subcontratado lo siguiente:

- Requisitos de SCM, incluido en el plan de SCM, el cual forma parte del acuerdo con los subcontratistas.
- Supervisión del subcontratista en cuanto a su cumplimiento.
- Revisiones y auditorías de configuración de los elementos del subcontratista que desempeñaran.
- Probar, verificar, aceptar y combinar con el software el código, documentación y datos externos.
- Cómo se procesarán los cambios, incorporando la participación del subcontratista.

Para el software adquirido, el plan detallará cómo se recibirá, probará bajo SCM el software, cómo se procesarán los cambios al software del suministrador y por ende, la participación del suministrador en el proceso de gestionar los cambios respectivamente.

1.3.5 Control de interacción

Las interacciones establecen una técnica apropiada para la construcción del software, el cual requiere habilidades en el desarrollo de proyectos, control y especificaciones de las interacciones, documentación del software, planificación de la gestión de configuración del software.

Para el control de las interacciones se establece un proceso a nivel de sistema, ocurre dentro del contexto del proceso, un elemento del software interacciona con otro elemento ya sea software o hardware, y produce cambios en el cual puede afectar cualquiera de los dos elementos, por tanto la planificación para los procesos identifica los elementos que interaccionan, cómo se producen estos cambios de dichos elementos como se comunicarán y gestionarán.

1.4 Plan de la gestión de configuración software

La planificación de SCM es vital para su éxito.

La planificación para un proyecto dado significa que será necesario planificar cómo se van a elaborar las actividades de SCM. Esto se refleja en un plan de gestión de la configuración, es un documento que se mantiene durante el desarrollo del ciclo de vida del software o sea se actualiza y aprueba, el plan es específico para cada proyecto siguiendo las pautas de los procedimientos y estándares del manual de calidad.

- Introducción: Se detalla el propósito, alcance y aplicación del plan, los términos que son claves así como la referencia.
- Objetivos de calidad del proyecto y enfoque adoptada para alcanzarlo.
- Documentación referenciada en el plan (manuales procedimientos etc.)
- Gestión de aseguramiento de la calidad (organización, actividades y responsabilidades).
- Documentación mínima exigida a los desarrolladores tanto del desarrollo del software (especificaciones, diseño, manuales de usuario, etc.) como de control (planes de validación y verificación).
- Estándares que se deben aplicar obligatoriamente.
- Actividades de revisión y auditoría.
- Gestión de la configuración del software (mediante un plan de gestión específico o describiendo sus actividades).
- Informes de problema, especificando la forma de tratar y corregir los problemas y los responsables que han de hacerlo.
- Herramientas para apoyar el aseguramiento de la calidad (revisiones, inspecciones, analizadores de código, generadores de entorno de prueba etc.) especificando sus objetivos y la manera de utilizarlas.
- Control de código (almacenamiento y versiones), control de acceso a equipos y prevenciones de seguridad.

- Recogida, almacenamiento y mantenimiento de datos sobre el aseguramiento de la calidad.

Tabla 37.Plan de gestión de configuración de software

El ámbito de aplicación será abordar las partes del SMC, las limitaciones y supuestos sobre los que se basa el plan.

Los siguientes elementos incluyen:

- Información general sobre la descripción del proyecto de software. - Identificación de los elementos de software a los que SMC se aplicará. - La identificación de otro software que se incluye como parte del plan (por ejemplo, el apoyo o el software de prueba). - Relación de SMC para las actividades de gestión de hardware o de configuración del sistema para el proyecto. - El grado de formalidad, parte del ciclo de vida del software para la aplicación de SCM en el proyecto. - Limitaciones, como la falta de tiempo, que se aplicará al plan. - Supuestos que podrían tener un impacto en el costo, horario, o la capacidad para realizar actividades definidas SMC (por ejemplo, los supuestos sobre el grado de participación del cliente en las actividades de SCM o la disponibilidad de ayudas automáticas).

Tabla 38.Supuestos y limitaciones de que consta el plan

1.5 Seguimiento de la gestión de la configuración del software

Después de que se haya implementado será necesario un seguimiento para asegurar que el proceso de la gestión de configuración del software es seguido de acuerdo a los procedimientos definidos como son los requisitos SQA.

Todo lo realizado se efectúe adecuadamente, esta tarea de seguimiento de la gestión de configuración del software ofrece la capacidad de seguir todos los componentes y entregables de diseño y construcción que resulte de una determina gestión de requisitos, esto puede suponer una actividad de cumplimiento de la auditoría de la garantía de la calidad del software, y asegura que se cumplan de manera correcta todos aquellos que tengan responsabilidades asignadas a cada una de las tareas SCM.

Para que cada una de las tareas se desarrolle de una manera fácil el empleo de herramientas integradas en la gestión de configuración proporcionan: procesos de control, el repositorio gestiona una amplia variedad de relación con los distintos objetos de la configuración, relación de entidades entre partes de diseño de la aplicación, componentes de diseño, otros productos de trabajo. Lo cual estas tareas mencionadas anteriormente es primordial para la integridad de la información y generación de productos. Estas aportaciones son muy importantes a la mejora del proceso del desarrollo del software.

1.5.1 Medidas y mediciones de la SCM

La medida asegura la disponibilidad de la información suficiente que permita el seguimiento acerca de la evolución del producto o una visión de cómo funciona el proceso. La utilización de un sistema de medición de procesos es una función primordial para controlar, administrar y mejorar los proyectos de manera eficaz. Las mediciones de procesos de la SCM proporcionan una efectividad de la SCM de las actividades de una manera continuada.

Las mediciones de procesos pueden incluir una de las siguientes recomendaciones:

a) Efectividad de la eliminación de defectos.
b) Control de transición de etapas.
c) Retraso del hito.
d) Rastreabilidad de requisitos.
e) Estabilidad de requisito y diseño.
f) Realización de pruebas.
g) Descubrir oportunidades para la mejora de procesos.
h) Monitorizan la efectividad de las actividades de SCM de una manera continuada.
i) Biblioteca del software.
j) Herramientas de SCM.

Podemos destacar los siguientes puntos relacionados con las necesidades de medición:

- Los modelos específicos de evaluación de procesos de software como CMM, suponen la clasificación de éstos mediante análisis subjetivos cuyo resultado suele ser un valor en una escala ordinal.

- ISO 9001[ISO, 2008] establece una serie de requisitos mínimos de medición que deben cumplir las organizaciones para obtener la calidad del proceso. Para la norma ISO 9001 para organizaciones del software, para la norma ISO 9000-3 corresponde con ciertos aspectos de la medición sobre técnicas estadísticas y sobre acciones correctoras y preventivas.

  En dicha norma se establece la necesidad de realizar mediciones de productos tanto características de calidad como intrínsecas del producto y de procesos tanto de cumplimiento de plazos y objetivos de calidad del proceso como de efectividad del proceso para reducir posibles defectos o para detectar lo que se produzca en cada proyecto.

  Hay que recoger valores e informar de ellos con regularidad, identificar el nivel actual de cada métrica, establecer objetivos específicos de mejora en función de las métricas y determinar acciones correctivas si los valores de métrica no alcanzan los objetivos previstos.

- Claridad de descripción.
- La relación entre el valor de la métrica y la calidad de proceso o producto tiene que estar justificada y ha de ser comprendida con facilidad.
- Hay que identificar la manera en que las técnicas de desarrollo pueden influir en el valor de la métrica.
- Se debe comprender cómo evoluciona el valor de la métrica a medida que mejora la calidad.
- Se tienen que describir las métricas que permiten comparar proyectos y proveedores.

Tabla 39: Principios útiles en el aseguramiento de la calidad del software en un proyecto

- En el caso de los modelos de mejora de proceso del software es importante estudiar los requisitos de medición que aparecen como consecuencia lógica de adoptar cualquier modelo. También hay que analizar cómo dichos requisitos deben cambiar a medida que el nivel de madurez del proceso se incrementa.

1.5.2 Auditorías durante el proceso de la SCM

La auditoría de configuración asegura que los elementos de configuración del software sean correctos, la documentación está actualizada y sea consistente con la versión que se ha construido.

Las auditorias son el medio por el cual una organización asegura que los miembros de desarrollo han realizado su trabajo de forma que cumplan con el compromiso externo. Las auditorías pueden cambiar en grado de formalidad y disciplina dependiendo del compromiso legal de cada organización implicada en la realización del software.

Una auditoría establece información acerca de cuándo, porque y por quien se efectuaron los cambios, estos cambios pueden introducirse como atributos de objetos específicos en el repositorio; en conclusión la auditoría durante el proceso evalúa el estado de los elementos específicos o del proceso a seguir.

El propósito del proceso de auditoría de software es determinar de forma independiente la conformidad de los productos seleccionados, los procesos con los requisitos, planes y acuerdos según el caso.

- Una estrategia de auditoría es desarrollada y aplicada.
- La conformidad de determinados productos de software de trabajo y/o servicios o procesos con los requisitos, los planes y acuerdos se determina según la estrategia de auditoría.
- Las auditorías se llevan a cabo por un organismo independiente apropiado.
- Los problemas detectados durante la auditoría son identificados y comunicados a los responsables de la acción correctiva, y resolución de resultados.

Tabla 40. Resultado del proceso de auditoría de software

2. Identificación de la Configuración del Software

La actividad de la identificación de la configuración del software controla las actividades de identificación de la configuración; identificarán, las descripciones documentadas como son: documentación física y funcional del código, especificaciones, diseño y elementos a controlar del software para controlar los respectivos cambios, esto supone entender la configuración del software en la configuración del sistema.

Los elementos controlados pueden ser productos intermedios o finales tales como:

- Código ejecutable y código fuente.
- Documentación de usuario.
- Enumeración de programas.
- Base de datos.
- Casos de prueba.
- Especificaciones.
- Plan de gestión.
- Elementos de entorno de soporte.

El plan de gestión de configuración identificará los elementos de configuración (SCI) del proyecto en cada parte de control del proyecto. Este plan establecerá como se nombra de manera única cada SCI y sus versiones y describirá las actividades a desarrollar para su definición, seguimiento, almacenamiento y recuperación de los elementos de configuración del software.

Tarea	Descripción
Identificación de los elementos de configuración y de la línea base	- Identificación de los elementos El plan SCM registrará los elementos a controlar, SCI del proyecto, y sus definiciones durante su evolución, se describirá como se mantendrán en el proyecto todos los elementos como sus estructuras, por lo menos se listarán aquellos SCI que serán considerados como entregables en el proyecto. - Línea base Deben definirse en las partes de control del ciclo de vida del proyecto estableciendo lo siguiente: 1. Evento que creará. 2. Elementos a controlar. 3. Procedimientos usados para el establecimiento y oportuno cambio. 4. Autoridad para aprobar los cambios a los documentos aprobados.
Nombre de los SCI	El plan SCM establecerá - Un sistema para especificar la asignación de los distintos identificadores a cada elemento a controlar. - Establecerá cómo debe identificarse de manera única cada versión de cada elemento. - Describirá los métodos para nombrar cada uno de los elementos controlados con el objetivo de almacenar, recuperar, seguimiento y distribución.
Obtención de los elementos de configuración	El plan SCM establecerá - Identificación de las bibliotecas software controladas por el proyecto. - Descripción de cómo se ubican físicamente bajo control en la biblioteca tanto el código como la documentación y los datos de la línea base. - Especificar procedimientos para el almacenamiento de documentos actuales, medios magnéticos incorporando el marcado y etiquetado físico de los elementos.

Tabla 41.Tareas necesarias para el desarrollo de la identificación de la configuración

2.1 Identificando los Elementos a Controlar

Se registrarán los elementos a controlar SCI del proyecto, se definirán a medida que evolucione. Se debe describir cómo se va a sustentar en el proyecto la lista de los elementos identificados y su estructura, por lo menos

deben listarse los elementos que serán considerados como entregables en el proyecto y podemos citar: Especificación de requisitos, el plan de gestión de configuración de los elementos(SCI), plan de proyecto.

En el sistema se estableció para la identificación de los elementos de software y sus versiones a ser controlados por el proyecto. Para cada elemento software y sus versiones, serán identificados mediante los siguientes aspectos:

- La documentación que establece la línea de base.
- Las referencias de la versión.
- Y otros detalles de identificación.

2.1.1 Configuración del software

El propósito del proceso de software de gestión de configuración es establecer y mantener la integridad de los elementos de software de un proceso o proyecto y ponerlos a disposición de las partes interesadas.

- En la gestión de configuración de software, la estrategia se desarrolla. - Los elementos generados por el proceso o proyecto se identifican, y se definen en la línea base. - Las modificaciones y versiones de los temas son controlados. - Las modificaciones y versiones están a disposición de las partes afectadas. - El estado de los elementos y las modificaciones se registran y comunican. - La integridad y coherencia de los elementos está garantizada. - El almacenamiento, manejo y entrega de los productos son controlados.

Tabla 42.Resultado de la implementación del proceso de gestión de configuración del software

2.1.2 Elemento de configuración del software (ECS)

Se define el elemento de configuración como una agregación de hardware y software, o de ambos, diseñada para la gestión de configuración y que se trata como una única entidad en el proceso de gestión de configuración.

Citamos algunos elementos de configuración:

- Especificación de requisitos del software. - SCMP. - Plan de proyecto. - SQAP. - Plan de pruebas del sistema. - Modelo funcional del sistema. - Diseño detallado. - Código fuente. - Casos de prueba y obtención de resultados.

Tabla 43.Algunos ejemplos de elementos de configuración del software

ECS es una información que se crea como parte de desarrollo de ingeniería del software. Son entidades básicas que permiten el progreso del software. Los ECS están organizados para formar elementos de configuración susceptibles

de catalogar en la base de datos del proyecto con un solo nombre. Los elementos tiene un nombre, atributos y están conectados con otros elementos por medio de relaciones.

El propósito del desarrollo del software es lograr conjuntos de elementos software interrelacionados, es decir, configuración esto representa el estado de realización del producto. Una configuración software está constituida por elementos de configuración aprobados en un momento del ciclo de vida del software. *La configuración facilita al cliente algo que puede comparar con sus necesidades en un momento dado del ciclo de desarrollo [BRYAN Y SIEGEL, 1984].*

2.1.2 Relaciones entre elementos de la configuración del software

Los elementos seleccionados en la relación estructural y sus partes, y pueden verse involucradas las actividades o tareas que puedan afectar a SCM.

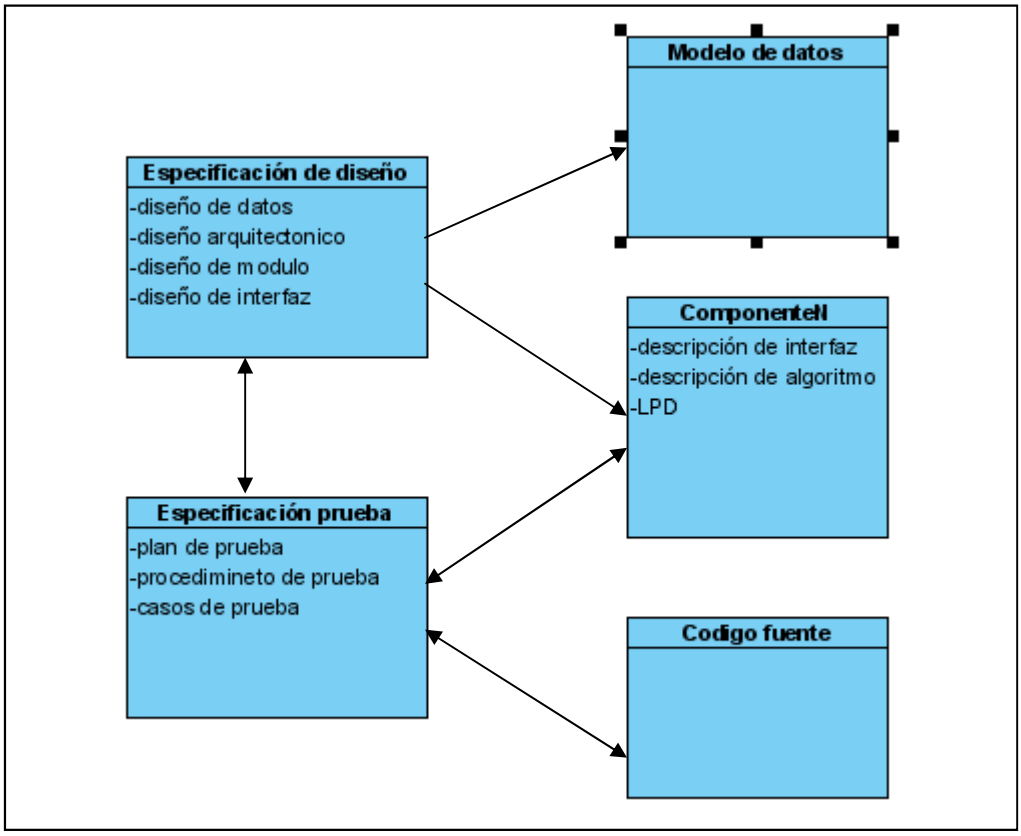


Figura 17.Relaciones de elementos de configuración

Un elemento de configuración tiene nombre y atributos y está relacionado con otros elementos. La especificación de diseño, Modelo de datos, especificación de pruebas, componenteN, código fuente están especificados por separado; pero están relacionados mediante flechas, una flecha doble significa interrelación. Si hubiese un cambio en el código fuente las interrelaciones permiten que un ingeniero de software determine qué ECS pueden afectarse.

2.1.3 Versiones del software

Conforme un proyecto evoluciona lo hacen al mismo tiempo los elementos del software.

Una versión de un elemento del software es un elemento identificado, debe definirse una forma no ambigua de identificar cada versión de los componentes para asegurar que se incluyan los componentes adecuados, no se puede usar el nombre del elemento de la configuración para identificar las versiones debido a que hay diferentes versiones para cada elemento de la configuración, existen tres técnicas básicas utilizadas para la identificación de los componentes.

1. Numeración de las versiones: Se asigna un número de versión claro y único.
2. Identificación basada en atributos: Cada componente tiene un nombre que es designado para los elementos de la configuración, que no es único a lo largo de las versiones, y un grupo asociado de atributos. Por lo tanto los componentes se identifican por su nombre y valor de los atributos
3. Identificación orientada al cambio: Cada versión del documento es creada en respuesta a una o más peticiones de cambio. La versión del sistema se identifica por el conjunto de modificaciones que se implementan en los componentes.

Un problema fundamental con los esquemas es la asignación de nombre de las versiones en el cual no reflejan los atributos distintos utilizados para identificar las versiones. Ejemplos de identificación de estos atributos son:

- El estado de desarrollo.
- El cliente.
- La plataforma de Hardware.
- La fecha de creación.

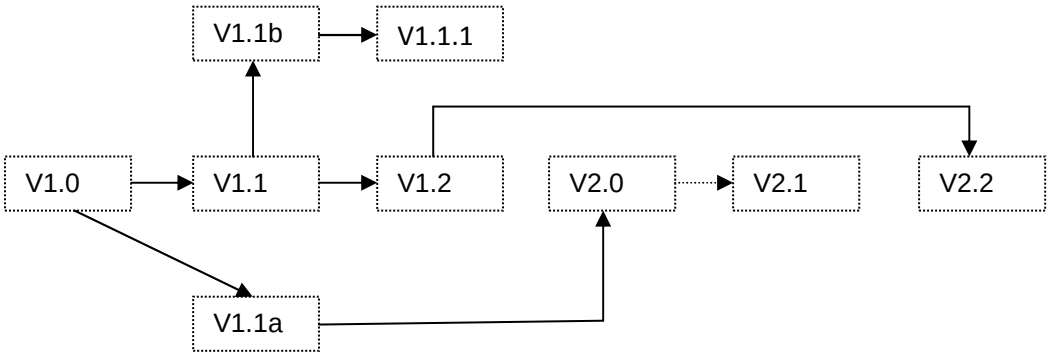


Figura 18. Estructura de derivaciones de las versiones. Adaptado de [Ian Sommerville]

Las flechas en este diagrama apuntan desde la versión fuente hasta una nueva versión del sistema creada a partir de la fuente. Observe que la derivación de versiones no es necesariamente lineal y las versiones con números de versión consecutivos se producen a partir de diferentes líneas base. En principio cualquier versión existente se utiliza como un punto de inicio para una nueva versión del sistema

"*Cualquier cambio, incluso para mejorar está acompañado con inconvenientes e incomodidades" Arnold Bennet*

2.1.4 Línea base

La importancia de la línea base se extiende más allá de la gestión de configuración la cual es una parte primordial en el desarrollo del software.

La definición más sencilla es la de Bryan y Siegel: "Una línea base es un fotografía aprobada del sistema en un punto dado de su evolución". Para el caso del software una línea base marca formalmente el final de una fase del ciclo de vida del software. Una línea base es el producto que emerge de un ciclo de desarrollo del software después de haber completado con éxito un proceso de revisión [BRYAN Y SIEGEL, 1984].

Podemos definir una línea base:

- Una línea base es algo permanente, puede modificarse por un proceso formal de control de cambio en el cual ya ha sido auditado.
- Una línea base permite resolver el desarrollo del software simplificando la consecución real para pasar desde la ideas al código. La línea base debe marcar un movimiento definitivo en el avance desde la creación inicial de los requisitos para el sistema hasta el software entregable.

La línea base simboliza el éxito de cada fase de desarrollo. En definitiva la línea base constituye hitos en el proceso de desarrollo. Especialmente la línea base viene definida por las fases del ciclo de vida seleccionada.

En la línea base por un lado se puede identificar cuáles son los productos de las distintas fases y por otro lado el aseguramiento que la fase se ha completado.

Pueden representar hitos en el desarrollo del software como puede ser:

Al finalizar cada etapa o fase del ciclo de vida para identificar productos de cada fase asegurando su terminación.

- Requisitos del sistema revisado.
- Especificación de requisitos revisada y las interfaces.
- Evolución del software durante el ciclo de vida del software.
- Autoridad de cambios.
- Productos de software completo entregado para la integración de sistemas.
- Las tareas de ingeniería del software producen uno o más elementos de configuración del software, se revisan y aprueban y se colocan en una base de datos del proyecto o llamada repositorio.

Etapa del ciclo de vida	Configuración de referencia que se produce	Componentes de una línea base
Análisis de requisitos	Línea base funcional	Línea base funcional: Se crea al terminar la fase de análisis y comprende los requisitos del sistema, SQAP,SCMP, plan de desarrollo plan de pruebas de aceptación
Diseño arquitectónico	Línea base de asignación	Línea base de asignación: se crea al terminar la fase de diseño arquitectónico y comprende la arquitectura del sistema, especificación de interfaces de subsistemas, plan de pruebas de sistema
Diseño detallado	Línea base de diseño	Línea base de diseño: se crea al terminar la fase de diseño detallado de los subsistemas, plan de pruebas de integración.
Implementación/integración	Línea base de producto	Línea base de producto: Se crea al terminar la codificación y las pruebas de integración y comprende: código fuente, objeto y ejecutable, resultado de las pruebas de integración. Versión de manuales de usuario
Plan de pruebas	Línea base de explotación	Línea base explotación: Se crea al terminar la implantación, se incluirán los resultados de prueba en el sistema, pruebas de aceptación y documentación de usuario
Mantenimiento/Explotación	Nueva línea base de explotación	Nueva línea base de explotación: con las mismas características de una línea base explotación

Tabla 44.Ciclo de vida del software y línea base

2.1.5 Adquisición de elementos de configuración del software

El plan debe especificar los procedimientos para el acopio actual de los documentos y los medios magnéticos, introduciendo el marcado y etiquetado físico de los elementos.

Describirá el periodo de mantenimiento de los datos, así como los procedimientos de preparación y reparación de desatres. Estos elementos de recuperación y reproducción controlados apartir del acopio de la biblioteca. Por

lo tanto el SCMP debe identificar las bibilotecas software controladas para el proyecto como se encuentran físicamente en la biblioteca tanto el código, documentación y los datos de LB identificadas.

Los ECS se encuentran bajo el control de la gestión de configuración del software en instantes diferentes; lo que se añade a una LB en instantes específicos del ciclo de vida del software. En la obtención de ECS, las modificaciones a los elementos se deben aprobar de manera apropiada para dicho elemento y LB involucrada; después de la aprobación se ingresa en la LB.

2.2 Biblioteca de Software

La biblioteca de software es una colección donde se almacena el software para ofrecer un adecuado manejo de código de software, documentación, medios de comunicación, manejo de versiones, datos relacionados en diversas formas, múltiples desarrolladores y está diseñada para ayudar en el desarrollo del software uso y mantenimiento.

Para cada biblioteca se deberá determinar el formato, los requisitos de documentación, requisitos de recepción e inspección, procedimientos de control de acceso, localización, etc. Las tareas mencionadas anteriormente deben ser efectuadas por miembros de la organización que tengan establecido el rol de bibliotecario.

En el proyecto pueden existir distintos tipos de biblioteca:

Biblioteca	Descripción
Biblioteca estática	Usada para archivar la LB libradas para uso general.
Biblioteca dinámica	Realizada por el propio trabajador y controlada por él.
Biblioteca maestra	Usada para resolver la línea base existente y controlar los cambios sobre ella, tiene fuertes restricciones de acceso para escritura, aunque no las tiene para lectura normalmente; debe ser autorizada por el comité de control de cambios.
Biblioteca de trabajo	Se determina al inicio del proyecto, donde el analista y diseñadores desarrollan los distintos documentos del proyecto, los programadores elaboran el software.
Biblioteca de soporte al proyecto	Se almacenan los elementos de configuración del software aprobado y trasladado desde la biblioteca de trabajo o biblioteca de integración. Cuando un elemento transita a la biblioteca de soporte este se encuentra ante un control de cambio interno.
Biblioteca de integración	A nivel superior del sistema a través de esta biblioteca se adquieren los elementos de configuración para su integración.
Repositorio del software	Es la entidad en la que archivan los ECS de un

	proyecto para su cierre.
Biblioteca Backup	Aunque su contenido no está impregnado en la gestión de configuración; pero sí debe estar identificado.

Tabla 45.Tipos de biblioteca

3. Control de la Configuración del Software

El control de configuración del software le corresponde durante el ciclo de vida del software a la gestión de cambios, el control de cambios es vital, son de gran importancia los cambios porque una pequeña transformación en el código puede crear un gran fallo en el producto, el cambio incontrolado conduce al caos, esta actividad es primordial y su principal objetivo es realizar un mecanismo para el control de cambios, combina procedimientos humanos y herramientas automatizadas.

Las actividades de control de la configuración piden, evalúan, aprueban o desaprueban e implementan los cambios de los ECS de la línea base estos cambios pueden ser tanto de mejora como corrección de errores. El grado en que se evalúe en el proceso depende de la línea base afectada en el proyecto y del impacto que supone este cambio en la estructura de configuración.

Para cada biblioteca software el SCMP debe describir el control de cambio que se ha establecido en ECS de la línea base definiendo las siguientes tareas:

- Identificación y documentación de la necesidad del cambio.
- Análisis y evaluación de la petición de cambio.
- Aprobación o desaprobación de la petición.
- Verificación, implementación y liberación del cambio.

Tabla 46.Tareas específicas en el plan de gestión de configuración del software

Además SCMP identificará los registros a usar para desarrollar el seguimiento y la documentación de la secuencia de pasos para realizar el cambio y todos los cambios basados en el origen de la petición debe ser documentada.

Tareas Control de la Configuración del Software

3.1 Petición, Evaluación y Aprobación de Cambios del Software

Tareas requeridas para desarrollar el control de la configuración:

- **Petición de cambio**: El SCMP especificará los procesos para solicitar un cambio de ECS de la línea base, la información a documentar para la petición. La información a inspeccionar para el cambio debe contener como mínimo la siguiente información.

1. El nombre y la versión de ECS donde aparece el inconveniente. 2. Nombre y organización del autor de la solicitud. 3. Fecha de solicitud. 4. Indicar la urgencia. 5. La necesidad de cambio. 6. Características del cambio solicitado. 7. Prioridad o clasificación para clasificar la solicitud y ayudar en su análisis y evaluación. 8. Número de solicitud de cambio estado y disposición. 9. Registrar el cambio para realizar el seguimiento de cambio.

Tabla 47.Información registrada para un cambio

- **Evaluación de los cambios**: En SCMP se deberá especificar el análisis requerido para diagnosticar el impacto del cambio suministrado y el proceso para comprobar los resultados del análisis.

  Los cambios deben desarrollarse de acuerdo a los entregables y el impacto en los recursos del proyecto. El jefe del proyecto o la persona que reemplace al jefe del proyecto analizarán todos los cambios pertinentes, además debe especificar los estándares de calidad que requiere la empresa.

- **Aprobación o desaprobación de los cambios:** SCMP debe identificar cada comité de control de cambio y su nivel de poder y responsabilidad definidos en SCM para aprobar los cambios establecidos.

Un comité de control de cambio puede determinar múltiples niveles de comité de control de cambio dependiendo del grado de complejidad y la línea base del proyecto y es la organización que afirma que los cambios se procesan de una manera visible y trazable y todos aquellos miembros implicados reciban la información oportuna.

El SCMP debe determinar el nivel de solicitud de cambio, incluyendo modificación durante el ciclo de vida.

3.1.1 Consejo de control de la configuración del software (CCB)

El CCB tiene la responsabilidad de aceptar o rechazar los cambios que se han establecidos.

3.1.2 Proceso de petición de cambios del software (SRC)

Un proceso de petición de cambio del software, la utilidad de herramientas de soporte que permita la recolección de datos e información asociada, definición de formularios que se deben emplear para la solicitud del cambio, mejoras o corrección de defectos.
Estos formularios deben contener una información que determine:

Pasos	Procedimientos
- Informe del problema. - Defecto de cambio, - Analiza el impacto y la exigencia. - Priorización (cambio de solicitud normal o urgente). - Requisito de atención.	- Aseguramiento de que se cumplan cada uno de las pautas establecidas así como su correspondiente registro. - Cuándo, quién, por qué, qué el cambió. - Características detallando el problema y mejoras de solución. - Elementos de configuración del software se pueden ver afectados por el cambio. - Aprobación. - Seguimiento de soluciones para informes de errores. - Etc.

Tabla 48.Formulario a emplear mediante pasos y procedimientos para la solicitud de cambio

Podemos utilizar distintos informes en los cuales nos mostrará el estado actual de las peticiones de cambio y el tiempo en el que se encuentren en un estado determinado y su evolución, entre los distintos informes cabe mencionar:

Informes	Descripción
Informes de incidencia	Recoge información de la existencia del problema
Informes de solicitud de cambio	La implementación se relaciona en el sistema mediante el diseño, pero la solvencia en el diseño solucionaría el problema que lo generó.
Informes de notificación de cambio	Puede dar respuesta a varios Informes de Incidencias, siempre y cuando corresponden al mismo problema. La deficiencia se describe en el informe por la inadecuada implementación que debe ser solventada.
Informes en la documentación	La deficiencia tiene algún impacto sobre la documentación.

Tabla 49.Informes de petición de cambio

```
Solicitar cambios completando un formulario de solicitud de cambio
Analizar la solicitud de cambio
If cambio es válido then
        Evaluar cómo implementar el cambio
        Evaluar los costos de cambio
        Registrar la petición del cambio en una base de datos
        Remitir la petición a la oficina de control de cambio
        If cambio es aceptado then
                Repeat
                        Hacer cambios al software
                        Registrar cambios y vincularlos a la petición de cambios
asociados
                        Remitir el software cambiado para aprobar la calidad
                Until  calidad del software sea adecuada
                Crear nueva versión del sistema
        Else
        Rechazar petición de cambios
Else
Rechazar petición de cambios
```

Figura 19.Proceso de petición de cambios del software. Adaptado de [Ian Sommerville]

3.2 Implementación de cambios en el Software

El SCMP deberá determinar las distintas actividades para verificar e implementar un cambio adecuado. La información para el desarrollo de un cambio como mínimo debe contener en su registro lo siguiente:

- Solicitud o solicitudes de cambio asociada.
- Nombres y versiones de los elementos a analizar.
- Fecha de comprobación y grupo responsable.
- Fecha de instalación y grupo responsable.
- Identificador de una nueva versión.
- Métricas de fallos del software.
- Identificación del software de soporte usado para implementar el cambio.
- Construcción de una versión para pruebas.

Tabla 50.Registro de la implementación de cambios en el software

Una vez aceptados los elementos de configuración del software, el jefe de configuración debe ser el responsable de la coordinación de las pruebas y de la integración de SCI modificados. Estos cambios deben desarrollarse según la documentación de pruebas de regresión explicadas en la parte de pruebas del software. El jefe de configuración debe acoplar la construcción de una versión para prueba.

Hacemos hincapié en el proceso de aseguramiento de la calidad para el proyecto que se establecerá. Los objetivos del proceso de aseguramiento de la calidad será asegurar que los productos de software y los procesos empleados para proporcionar los productos de software cumplen con los requisitos establecidos y se adhieren a sus planes establecidos.

3.3 Desviaciones y Remisiones

- Tamaño. - Complejidad. - Gestión del proyecto y la organización. - Análisis de coste y beneficio. - Necesidades que no pueden ser cubiertas durante el desarrollo del ciclo de vida del software. - Impacto sobre el rendimiento, utilidad, requisitos, etc. - Recursos disponibles tanto humanos como materiales para realizar el cambio - Tiempo de respuesta para completar el cambio. - Utilización de algún elemento después de su desarrollo. - Políticas de la empresa (cumplimiento de objetivos entre los cuales podemos. mencionar: competitividad, meta, satisfacción del cliente etc.) - Existencia de alternativas buscando siempre mejoras de desarrollo.

Tabla 51. Factores que determinan la decisión de tomar o rechazar las solicitudes de cambio

4. Registro del Estado de la Configuración del Software (IEC)

El informe de estado, denominado también contabilidad de estado. El éxito del proyecto en el desarrollo del software es la generación de informes de estado en el cual juega un papel fundamental, ya que su objetivo primordial es mantener a los gestores, usuarios y desarrolladores alertas a los cambios y su evolución.

La generación de IEC es una tarea SCM que pretende dar respuestas a las siguientes preguntas:

1. ¿Qué ocurrió?
2. ¿Quién lo hizo?
3. ¿Cuándo ocurrió?
4. ¿Qué cosa será afectada?

La contabilidad de estado de la configuración provee medios para determinar dónde nos encontramos y como se están desarrollando las actividades.
Mejora los problemas de comunicación entre las personas involucradas en el proyecto; para solventar la falta de comunicación entre los participantes se registra la información y se genera informe.

Se destacan tres tareas fundamentales:

Información → almacenamiento → informe

4.1 Información del Estado de la Configuración del Software

El SCMP debe incorporar información acerca de:

1. Qué elementos de datos van a seguirse e informarse en la línea base y su correspondiente cambio.
2. Qué tipo de informes y registros de estado de la configuración se van a generar y con qué frecuencia.
3. Como se va a recoger, almacenar, procesar e informar la información.
4. Como se va a controlar el acceso a los datos de estado.

Tabla 52.Información que se debe incluir SCMP

Para ECS deben seguirse e informarse como mínimo los siguientes elementos de datos:

1. Versión inicial aprobada.
2. Estado de los cambios solicitados.
3. Estado de la implementación de los cambios aprobados.
4. Nivel de detalle y los datos pueden cambiar de acuerdo a las necesidades de información por parte del cliente como del proyecto.

Tabla 53.Información que se debe incluir ECS

La contabilidad del estado de la configuración tiene como propósito implementar los procedimientos para gestionar la información que se produce en el proceso de desarrollo con el propósito de producir una imagen eficiente sobre el estado del producto, proyecto y la evolución del producto.

Es necesario mantener la continuación en el desarrollo del software ya que los miembros implicados pueden abandonar su puesto de trabajo, por lo tanto es de gran importancia el historial de proyecto ya que contiene la experiencia lograda durante el desarrollo. Esta experiencia lograda prevé la oportunidad de evitar errores anteriores.

"Los beneficios de una contabilidad del estado de la configuración eficaz son el ahorro de tiempo y dinero que supone hacer las mismas cosas una y otra vez, sólo porque nadie sabe que ya se han hecho" [BERSOFF et al. 1980].

4.2 Informes del Estado de la Configuración del Software

Los informes pueden ser utilizados por los participantes del proyecto, ya sea por el equipo de desarrollo, mantenimiento, por las distintas actividades, gestión del proyecto etc. Los informes generan durante el ciclo de vida del software registros de garantía de calidad.

Informes	Registros
- Informe estado de cambio. - Resumen de las peticiones de cambio y estado. - Resumen de los cambios realizados a la línea base y su frecuencia. - Resultado de la auditorías de la línea base. - Informe de incidencias. - Informe de modificaciones. - Informe de distintas versiones.	- Registros de elementos de configuración. - Registros de línea base. - Registro de solicitud de cambio. - Registro de cambios. - Registro de versiones. - Registro de incidencias. - Registro de modificaciones de código. - Registro de modificaciones sobre base de datos. - Registro de modificaciones sobre documento. - Registro de variantes. - Registro de instalaciones. - Actas de reuniones de comité de control de cambios.

Tabla 54.Informes y registros del estado de configuración del software

En cada proyecto es necesario establecer los registros y los informes que se van a generar y para quién.

5. Auditoría de la Configuración del Software

Las auditorías son el medio por el cual una organización afirma que el personal de desarrollo ha realizado con éxito su trabajo cumpliendo con las expectativas externas. Se verifica si el producto está completo identificando cada problema y verificando su respectiva corrección.

La auditoría de configuración del software es una de las actividades más costosas en tiempo y recurso de la gestión de configuración del software, las causas son: dificultad y experiencia por parte de los auditores en desarrollar las tareas, se requiere un número de personas que realizan distintas tareas en un tiempo limitado. En el cual se requiere personal cualificado y con conocimiento en el proceso de desarrollo.

Herramientas que dan soporte a la planificación y ejecución. Una auditoría software eficaz implica un nivel técnico y una gran experiencia como se aplica en la gestión de desarrollo. Por lo tanto el desempeño de los auditores con experiencia es costoso, pero este coste es necesario si se quiere alcanzar seguridad en la calidad del producto.

La gestión de configuración del software en cuanto a auditoria de la configuración contempla:

- La auditoría de configuración es una función de soporte que favorece los elementos para todas las auditorías del desarrollo y certifica que los elementos auditados están completos y siguen las respectivas normas de SCM. - La auditoría de configuración se desarrolla para certificar que la línea base de producto se corresponde con las características del elemento en la especificación y documentación.

Tabla 55. Auditoría de la gestión de configuración

En SCMP deberá definir para cada auditoría o revisión de la configuración lo siguiente:

- Objetivos específicos. - Los ECS a auditar. - Cronograma de las tareas de auditoría. - Procedimientos que se llevan a cabo en la auditoría. - Integrantes que van a participar en la auditoría. - Documentación solicitada en la revisión o auditoría. - Procedimiento para registrar deficiencias e informar de acciones correctivas. - Criterios de aceptación y las acciones a realizar después de la aceptación.

Tabla 56.Definición para cada auditoría de la configuración

Tareas de Auditoría de la configuración del software

5.1 Auditoría de la Configuración Funcional del Software

Su objetivo es garantizar consistencia con la especificación de elementos de configuración del software que se están auditando, obteniendo resultados como entradas de datos clave en la verificación y validación del software realizada.

5.2 Auditoría de la Configuración Física del Software

Su objetivo es garantizar que el diseño y la documentación prescrita sean consistentes con el producto del software tal y como se ha construido es decir representación del software codificado y aprobado.

5.3 Auditorías durante el proceso de una línea base de software

Determinar el estado actual de ECS de la configuración que se lleva a cabo mediante el proceso de desarrollo del software asegura la consistencia y rendimiento de la línea base según su especificación o puede garantizar que la documentación es consistente con el elemento de configuración del software de la línea base que en ese momento se está desarrollando.

6. Gestión del Lanzamiento y Distribución del Software

Lanzamiento: Distribución de un elemento de la configuración de software que se encuentran fuera de las actividades de desarrollo. Pueden ser internas al proyecto así como la distribución del cliente a la hora de su entrega.
La biblioteca de software es un componente primordial para desarrollar las tareas de lanzamiento y distribución del software.

Tareas de Gestión del Lanzamiento y Distribución del software

6.1 Construcción del Software

La construcción del sistema es el proceso de vincular los componentes o elementos del software y compilar un programa que se ejecute en un programa en particular.

Se especifican las herramientas usadas para compilar y vincular los elementos del software, esta dependencia entre los elementos se especifica en la secuencia de comandos de construcción, en la cual el sistema de construcción de cada elemento debe compilarse y cuando se puede, reutilizar el código del objeto existente. Para la generación de ejecutables nos tenemos que realizar las siguientes preguntas:

1. ¿Se encuentran todas las dependencias de vinculación y compilación desarrolladas adecuadamente que se incluyen en las instrucciones de la construcción? 2. ¿Las versiones de cada elemento que se incluyen son las adecuadas? 3. ¿Se encuentran todos los archivos de datos disponibles? 4. ¿Los archivos de datos están relacionados con los elementos ya que su nombre que se usa debe ser igual al nombre de los archivos de datos de la máquina donde se ejecutara el software? 5. ¿Disponibilidad de las versiones del compilador y herramientas que se requieran, ya que puede haber incompatibilidad con las herramientas de versiones anteriores utilizadas para el desarrollo del software con las herramientas que se están utilizando actualmente? 6. ¿Sistemas de tamaño grandes la compilación y vinculación de sus elementos puede con llevar a varias horas?

Tabla 57.Desarrollo de preguntas para la generación de ejecutables

Creación ejecutable del producto

Es una actividad de combinar las versiones correctas de los ítems de gestión de configuración del software, en un programa ejecutable para su distribución. El producto se construye utilizando versiones particulares de la herramienta de soporte por ejemplo compiladores y automatización.

El proceso y el producto de la construcción están ligados para la verificación y calidad del software, los resultados obtenidos en el proceso de la construcción

se podrían utilizar para futuras referencias y convertirse en registro de garantía del producto.

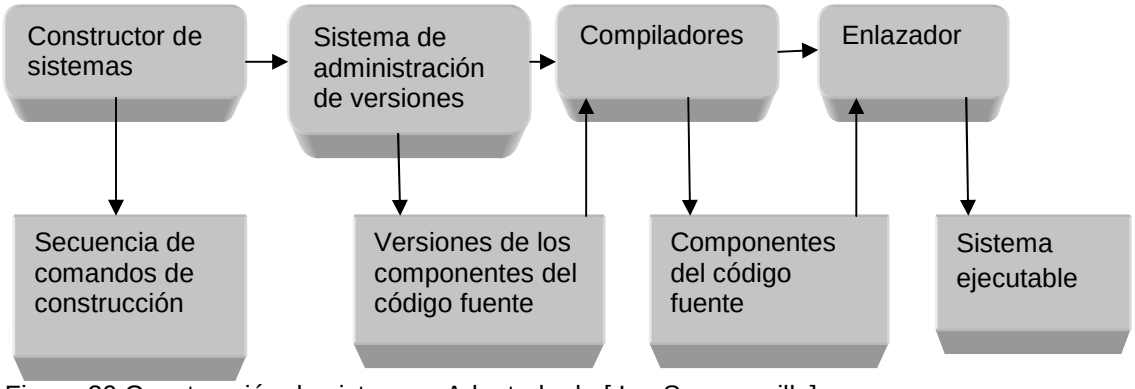


Figura 20.Construcción de sistemas. Adaptado de [Ian Sommerville]

6.2 Gestión del Lanzamiento del Software

La gestión del lanzamiento del software involucra la identificación, empaquetamiento, lanzamiento de elementos del producto es decir:

- Documentación, ejecutables, datos de configuración.
- Dirección de web para su correspondiente descarga, dispositivo de almacenamiento etc.

El empaquetamiento determina las iteraciones del producto, las instrucciones de modificación, actualización, selecciona qué variantes son fundamentales etc.

Las herramientas de soporte como son las herramientas CASE son esenciales para los procesos de configuración, de gestión de configuración están estandarizados y procedimientos predefinidos en las aplicaciones.

Se manejan grandes cantidades de datos y usuarios. Cuando se construye un sistema a partir de los componentes, un error en la gestión de configuración implica que el software no realice lo que se espera y no trabaja de forma adecuada.

Estas herramientas pueden crear entornos de trabajo en la gestión de configuración según distintas tareas.

- Entornos de trabajo abiertos: Integración de acuerdo a procedimientos estándares, como ejemplo tenemos:
    - Gestion de cambios lleva a cambio herramientas de seguimiento *(bug-tracking) como bugzilla.*
    - Gestión de versiones como *SCR (Tichy, 1985)*
    - *CVS (Berliner,1990)*
    - Construcción del sistema con herramientas como *Make (Feldman, 1979; Oram y Talbott, 1991)*
    - *Imake (Dubois, 1996)*

- Entornos Integrados cerrados: ofrece integración para la gestión de versiones, construcción del sistema, seguimiento de cambios es decir son complejos y además de coste muy elevado, por tanto se emplean herramientas individuales simples y económicas, como ejemplo tenemos:

    - Proceso de gestión de Cambios unificado de *Rational integrado que incorpora CleatCase (White, 2000) para la construcción y Gestión de versiones.*
    - *Seguimiento de cambios ClearQuest*

    La ventaja de los entornos integrados es que el intercambio de datos es muy sencillo y posee un entorno de bases de datos en la gestión de configuración integradas.

- Gestión de cambios: Una vez que se ha realizado la gestión de cambios, la naturaleza del procedimiento significa un cambio en el modelo de proceso con un sistema de gestión de versiones en el diseño e integración como ejemplo de herramientas podemos mencionar:

    - Herramientas simples, open-source como Bugzilla
    - Herramientas integradas, Rational ClearQuest.

    Estas herramientas de gestión de cambios poseen las siguientes características:

    1. Editor de formulario: Cambiar los formularios que se van a crear y rellenar por aquellos que hacen las peticiones de cambio.
    2. Sistema de flujo de trabajo: Permite procesar las solicitudes de petición de cambio y el orden de procesamiento.
    3. Base de datos de cambios: Permite gestionar todas las propuestas de cambio y aquellas que se pueden vincular al sistema de gestión de versiones.
    4. Gestión de informe de cambio: permite generar informes sobre el estado de petición de cambio.
    5. Aprobación de cambio.

- Gestión de versiones: Controla un repositorio en el cual su contenido no cambia, para el trabajo de un elemento de configuración se extrae del repositorio y se crea un directorio de trabajo, después de realizar los cambios oportunos se introducirá de nuevo en el repositorio, la gestión de versiones posee las siguientes características : repositorio de almacenamientos de ítems, gestión de almacenamiento (versiones basadas en delta), registro de historial de campo, desarrollo independiente, conflicto de versiones, apoyo al proyecto etc.

- Generación de ejecutables y construcción: Compiladores, lenguaje, compilación distribuida, manejo de datos, podemos citar un ejemplo los entornos de desarrollo de Java, el script para construir el sistema lo primero que hace es crearse automáticamente analizando el código fuente y los componentes utilizados.

4.1.4.5 Estándares, prácticas, convenciones y métricas

Detalle de lo que está acordado y establecido para el proceso y los productos a obtener.

4.1.4.5.1 Propósito

En esta sección se deberá:

a.) Identificar las normas, prácticas, convenciones, las técnicas estadísticas a utilizar, los requisitos de calidad, y las métricas que se aplicarán.
b.) Estado de conformidad con estos componentes debe ser supervisada y garantizada.

Estas tienen el fin de asegurar la calidad pero también datos de métricas cuantitativas en el proceso de aseguramiento de la calidad del software. Además, estos datos se utilizarán para ayudar a elevar el nivel CMM, del nivel 3 al nivel 4 que los desarrolladores realizarán.

4.1.4.5.2 Contenido

Los temas tratados incluyen el diseño básico de técnicas, de programación y las actividades involucradas, tales como documentación, variables y nombres de módulo, programación, inspección y pruebas.

Estándares

Los procesos del aseguramiento de la calidad del software definen estándares para los procesos de desarrollo y producto del software, para lo cual los procesos se apoyan en herramientas que se establecen en los estándares de la calidad, para ello existen dos tipos de estándares que están relacionados en la gestión de calidad:

1. Estándares del proceso: Definen los procesos durante el desarrollo del software como son: especificación, diseño, validación y detalle de los documentos que se originan en la duración de los procesos.
2. Estándares del producto: Se aplican al desarrollo del producto software incluyen:

Estándares del producto	Descripción
Estándares de documentación	Son documentos de esencial importancia ya que representa el software y al proceso del software en forma tangible. Son fáciles de leer y entender ya que representan una estructura y calidad sólida.
Estándares de diseño	Utilizar un estándar para el diseño del programa tanto para el diseño arquitectónico como diseño detallado.
Estándares de codificación	Como utilizar un lenguaje de programación.
Estándares de comentarios	Para utilizan el programa usar comentarios en el inicio de cada rutina que indique el propósito, limitaciones evitando comentarios recargados explicando la idea principal de la realización de cada componente.

Tabla 58.Estándares de producto

El equipo que desarrolla los estándares se basa en estándares nacionales e internacionales. Usan los estándares como punto de arranque. A continuación describiremos tanto los estándares de producto como de proceso:

Estándares	Descripción
Estándares del producto	- Manual de revisión del diseño. - Organización del documento de requisitos. - Forma de solicitud de cambios. - Formato de la síntesis del proyecto. - Estilo de programación. - Formato del encabezado de procedimientos.
Estándares del proceso	- Canal de revisión del diseño. - Proceso de proporcionar las versiones. - Desarrollo de control de cambio. - Desarrollo de registro de las pruebas. - Desarrollo de la aprobación para el plan del proyecto.

Tabla 59.Estándares a nivel internacional de proceso y producto

Prácticas

- Retrasar la calidad es costoso. - Ninguna función del aseguramiento de la calidad separada que sea externa a cada personal destinado a realizar dicha función, puede en sí obtener una calidad mayor que la del desarrollador. - Todos los elementos del proyecto se inspeccionan y se pondrán a disposición del equipo una vez que los libere el desarrollador. - Todos los procesos deben revisarse por lo menos una vez para su correcto mejoramiento.

Tabla 60.Prácticas del producto

Convenciones

Cuando sea factible, el documento de las convenciones debe cumplirse con las sugerencias dadas en: *Escritura de ciencias de la computación: el arte de la comunicación efectiva de Justin Zobel (Springer Verlag: ISBN 9813083220) Referencia tomada del libro "Ingeniería del Software una Perspectiva Orientada a Objeto" por: Eric J. Braude.*

Métricas de la calidad del software

"No se puede controlar lo que no se puede medir" Tom De Marco (1982).

Se describen métricas de calidad como una categoría de herramientas SQA:

- Una medida cuantitativa del grado en que un elemento tiene un atributo de calidad determinado.
- Una de las funciones cuyas entradas son software de datos y cuya salida es un único valor numérico que se puede interpretar como el grado en que el software posee un atributo de calidad determinado. Es decir se refiere al proceso que produce las métricas de calidad.

Los principales objetivos de la métrica de calidad:

1. Para facilitar un control de gestión, así como la planificación y ejecución de las intervenciones de gestión adecuadas. El logro de estos objetivos se basa en el cálculo de las métricas con respecto a:

    - Desviaciones del actual funcionamiento de la calidad, los resultados del rendimiento planeado.
    - Desviaciones del cronograma actual y ejecución del presupuesto actual planificado.

2. Identificar situaciones que requieren mejora de los procesos de mantenimiento mediante medidas preventivas o correctoras introducidas en la organización. Este logro de objetivo se basa en.

    - Almacenamiento de información de métricas sobre el desempeño de unidades, equipos, estructura etc.

Clasificación de las métricas de calidad de software

Se utiliza un sistema de dos niveles.

1. Distingue entre el ciclo de vida y otras etapas del sistema de software:

    - Métrica de proceso: asociado con el procedimiento de desarrollo del software.
    - Métrica de producto: Asociado con el mantenimiento del producto.

2. Hace referencia a los términos de medir:

   - Calidad.
   - Cronograma.
   - Efectividad (de eliminación de errores y mantenimiento).
   - Productividad.

Las métricas de calidad de software implican una de las dos medidas siguientes para el tamaño del sistema:

- KLOC: Esta medida es específica para el lenguaje de programación o herramienta de desarrollo utilizado. Mide el tamaño del software en miles de líneas de código.
- Puntos de función: Una medida del desarrollo de los recursos (recursos humanos) necesarios para desarrollar un programa, basado en la funcionalidad especificada para el sistema.

Métrica de proceso

El procedimiento de métrica de software de desarrollo puede realizarse en una de las siguientes categorías:

1. **Métrica de procesos de software de calidad**

   La métrica de procesos de software de calidad se clasifica en tres clases:

   - **Error indicador de la densidad:**

Código	Nombre	Fórmula de cálculo
CED	Código de la densidad de error	CED =NEC/KLOC
DED	Desarrollo de la densidad de error	DED=NDE/KLOC
WCED	Código de error ponderado de densidad	WCED=WCE/KLOC
WCEF	Ponderado código errores por punto de función	WCEF/ NFP
WDEF	Ponderado de desarrollo errores por punto de función	WCEF/ NFP

Tabla 61.Error indicador de la densidad

Clave

NCE = número de errores de código detectados en el código del software por código de las inspecciones y pruebas. Los datos de esta medida son extraídos de los informes de inspección de código y pruebas.

KLOC = miles de línea de código.

ECM = número total de errores de diseño y código detectados en el proceso de desarrollo de software.

WCE = errores detectados código ponderado. Las fuentes de datos para este indicador son las mismas que las de NCE.

WDE = número total de errores de diseño y código detectados en el desarrollo del software. Las fuentes de datos para este indicador son las mismas que las de ECM.

NFP = número de puntos de función necesarios para el desarrollo del software.

- **Métrica error de gravedad**

Las métricas que pertenecen a este grupo se utilizan para detectar situaciones adversas de un creciente número de errores graves en situaciones donde los errores y los errores ponderados, medida por indicadores de densidad de error, están disminuyendo en general.

Código	Nombre	fórmula de cálculo
ASCE	Promedio gravedad de los errores de código	ASCE=WCE/NCE
ASDE	Promedio gravedad de los errores de Desarrollo	ASDE=WDE/NDE

Tabla 62.Métricas error de gravedad

- **Eficacia de la eliminación de error**

Los desarrolladores de software pueden medir la eficacia de la eliminación de error mediante el sistema de software de control de calidad después de un período de funcionamiento normal.

Código	Nombre	Fórmula de cálculo
DERE	Desarrollo errores eficacia de eliminación	DERE=NDE/NDE+NYF
DWERE	Desarrollo errores ponderado eficacia de eliminación	DWERE=WDE/WDE+WYF

Tabla 63.Error eficacia eliminación

Clave

WYF = número ponderado de los fallos de software detectado durante un año de servicio de mantenimiento.

2. **Métrica procesos de software cronograma**

   Un enfoque alternativo calcula el retraso medio en la terminación de los hitos. Los hitos se asignan para la realización en la etapa de plan del proyecto considerado en el cálculo de las métricas

3. **Métrica procedimiento del software de productividad**

   Este grupo de indicadores incluye:

   - Directa: Indicadores que tienen que ver con la productividad humana de un proyecto de recursos,
   - Indirecta: Las métricas que se centran en el grado de reutilización del software. Reutilización de software afecta sustancialmente la productividad y la eficacia.
   - Un plazo adicional: La evaluación comparativa de la productividad de desarrollo de software.

Métrica de producto

El conjunto de métrica de producto se clasifica de la siguiente manera:

1. **Métrica de calidad HD:**

   Los tipos de métricas de calidad HD se refieren a:

   - **HD llamadas densidad:**

     Se describen diferentes tipos de métrica. Algunos se refieren al número de errores y otros a un número ponderado de los errores. En cuanto a las medidas de tamaño / volumen del software, algunos utilizan el número de líneas de código mientras que otros aplican a los puntos de función.

   - **La gravedad de las cuestiones planteadas HD**

     Los resultados de la simulación pueden contribuir a las mejoras en todo o en parte a la interfaz de usuario facilidad de uso, así como el manual de usuario y menús de ayuda integrada.

Código	Nombre	Fórmula de cálculo
HDD	HD llamadas Densidad	HDD = NHYC / KLMC
WHDD	Ponderado HD llamadas Densidad	WHDD = WHYC / KLMC
WHDF	Ponderado HD llamadas al punto de función	WHDF = WHYC / NMFP

Tabla 64.HD llamadas métricas de densidad

Clave

NHYC = número de HD llamadas durante un año de servicio.

KLMC = miles de líneas de código de software.

WHYC = ponderada HD llamadas recibidas durante un año de servicio.

NMFP = número de puntos de función para mantenerse.

- **El éxito de HD**

  La medida más común para el éxito de los servicios HD es la capacidad para resolver problemas planteados por las llamadas del cliente en el plazo determinado en el contrato de servicio. Por lo tanto, la métrica para el éxito de los servicios de HD compara lo actual con el tiempo designado para la prestación de estos servicios.

2. Métricas de la productividad y la eficacia HD

Métricas de la productividad se refieren al total de recursos invertidos durante un período determinado, mientras que la eficacia de las métricas relacionadas con los recursos invertidos en la respuesta a una llamada al cliente de HD.

- **HD métricas de la productividad**

  Métricas de la productividad HD hacen uso de la facilidad de aplicar medidas KLOC del tamaño del sistema de mantenimiento del software o de acuerdo a la evaluación de puntos de función del sistema de software.

- **HD métricas de eficacia**

  La métrica de este grupo se refiere a los recursos invertidos en la respuesta a las llamadas de los clientes HD. Una métrica común es el que aquí se presenta, HD Eficacia (HDE): HDE = HDYH / NHYC

Código	Nombre	Fórmula de cálculo
HDP	HDP = HDYH / KLMC	HD Productividad
FHDP	Función de punto de alta definición de la productividad	FHDP = HDYH / NMFP

Tabla 65.HD métricas de la productividad

Clave

HDYH = total de horas anuales de trabajo invertido en el servicio.

HD del sistema de software.

3. Correctiva métrica de calidad de mantenimiento

Métricas de software de mantenimiento correctivo frente a varios aspectos de la calidad de los servicios de mantenimiento. Se requiere una distinción entre los fallos del sistema de software tratados por los equipos de mantenimiento y fallas del servicio de mantenimiento. Por lo tanto, las métricas del software de mantenimiento se clasifican de la siguiente manera:

- Software de fallos del sistema de medición de densidad

  Frente a la magnitud de la demanda de mantenimiento correctivo, en base a los registros de las fallas detectadas durante las operaciones regulares del sistema de software.

- Software de fallos del sistema de métricas de la gravedad

  Hacer frente a la gravedad de los fallos del software del sistema atendido por el equipo de mantenimiento correctivo.

4. Mantenimiento correctivo de la productividad y la eficacia de métrica

Un sistema de mantenimiento de software que muestra una mayor productividad requiere menos recursos para su tarea de mantenimiento, mientras que un sistema de mantenimiento de software más eficaz requerirá menos recursos, en promedio, para la corrección de un error.

Nombre	Código	Fórmula de cálculo
CMaiP	La productividad de Mantenimiento Correctivo	CMaiP = CMaiP / KLOC
FCMP	Función de punto de Mantenimiento Correctivo Productividad	FCMP = CMaiP / NMFP
CMaiE	Eficacia de Mantenimiento Correctivo	CMaiP = CMaiP / NYF

Tabla 66.Métricas de la productividad del software de mantenimiento correctivo y eficacia

Clave

CMaiYH = total anual de horas de trabajo invertido en el mantenimiento correctivo del sistema de software.

KLOC = miles de líneas de código de software.

NMFP = número de puntos de función designados para el software de mantenimiento.

NPV = número de fallos de software detectado durante un año de Servicio de mantenimiento.

4.1.4.6 Revisiones y auditorías

Las revisiones y auditorías se pueden usar para examinar tanto procedimientos de gestión del proyecto como productos. Se trata de documentos que se comunican o entregan al cliente o al equipo de desarrollo que se produce o adquiere durante el desarrollo o mantenimiento del software.

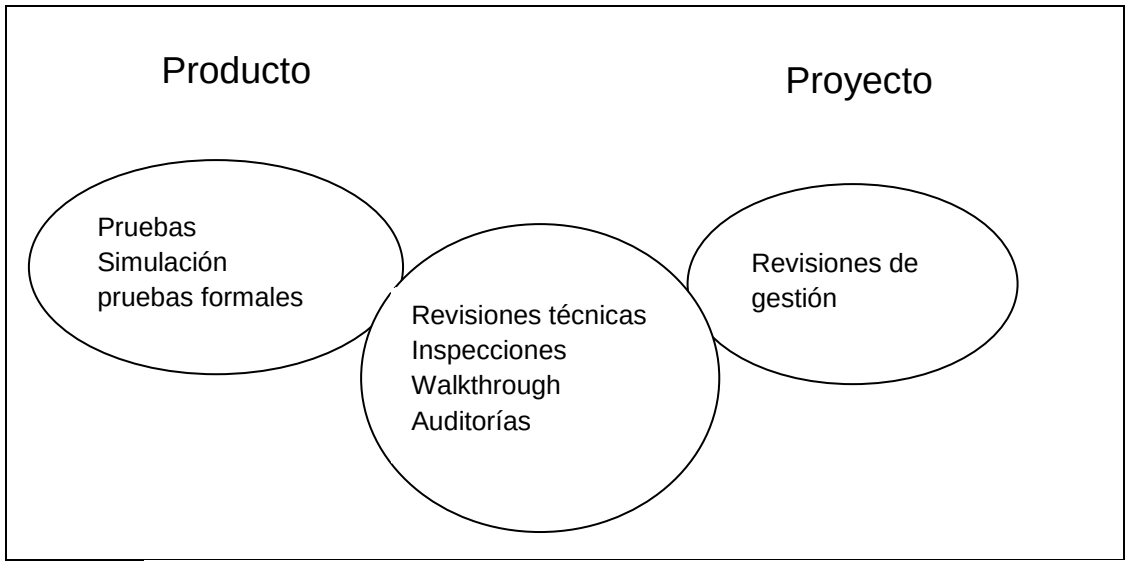


Figura 21.Relación de algunos procesos de aseguramiento de la calidad sobre productos y proyectos. Adaptado de [Mario G. Piattini]

Revisiones de Proyecto: Revisiones de gestión.

Revisiones de producto: Revisiones técnicas, inspecciones, walkthroughs, auditorías.

Hay que establecer las auditorías de software con las revisiones.

Atributo	Revisiones	Auditorías
Mecanismo	Las reuniones	Combinación de reuniones, observaciones y exámenes
Responsabilidad	Compartida entre un grupo de personas que se encuentra en la organización	Desarrollada por un grupo de personas que no pertenecen a la organización. Su principal figura es el auditor.
Duración	Corta	De media a larga
Anidamiento	Puede tener muchas secciones	Incluye otras auditorías, revisiones, e incluso pruebas periódicas
Frecuencia	Depende de la fase del ciclo de vida	periódica

Tabla 67.Diferencia entre las revisiones y las auditorías del software**.** Adaptado de [Mario G. Piattini]

4.1.4.6.1 Propósito

Los propósitos de las revisiones y auditorías son proporcionar un medio para centrar la atención en la calidad de la aplicación conforme se desarrolle. Las revisiones se llevan a cabo de manera exhaustiva y programada. Las auditorias se realizan con base de muestreos aleatorios sin previos avisos.

- Definir revisiones que se realizaron al software, pueden ser: revisiones técnicas, inspecciones, recorridos, auditorías etc.
- Listado de análisis de programas en el proyecto del software a través de cronogramas.
- Logro de análisis de programas.
- Medidas que se llevan a cabo y verifican.

4.1.4.6.2 Requisitos mínimos

Como mínimo, el análisis de programas siguientes se llevará a cabo.
SOFTWARE DE PLANES DE ASEGURAMIENTO DE LA CALIDAD Std. 730-2002.

4.1.4.6.2.1Especificaciones de software de revisión

El RSS se lleva a cabo para asegurar la adecuación de los requisitos establecidos en el DRS.

La revisión de los requisitos de software se realiza para asegurar la consistencia entre la definición del sistema, el plan de proyecto, la especificación de los requisitos para la producción del software, el plan de verificación del software y el manual de usuario.

La revisión de requisitos se realiza de forma manual, involucran a personas tanto de la organización como el cliente. Estas personas tienen la función de verificar la documentación de requisitos en cuanto se refiere a incompatibilidades, anomalías, defectos etc.

Las revisiones de los requisitos pueden ser:

- Revisión informal*: Los contratistas deben tratar con tantos stakeholders del sistema "Aquellos que tienen, reclaman, o poseen, derechos o intereses en una organización y en sus actividades" (Clarkson, 1995).*

  No hay procedimientos definidos. La revisión se realiza de la forma más manejable posible. Estas revisiones acarrean ciertas ventajas: menor coste, menor esfuerzo, preparación reducida, etc. En contrapartida detectan menos defectos.

- Revisiones semi-formales: Definen unos procedimientos a seguir los cuales son mínimos. Ejemplo de estas revisiones son walkthroughs, su objetivo es la evaluación de un producto para:

- Buscar defectos, omisiones y contraindicaciones. - Mejorar el producto. - Evaluación de conformidad con estándares . - Considerar posibles soluciones y alternativas en los problemas que se han encontrado.

Tabla 68.Revisiones walkthroughs

- Revisión formal: El equipo de diseño debe guiar al cliente a través de los requisitos del sistema, explicándoles la responsabilidad de cada requisito. La verificación de cada requisito debe verificarse por parte del equipo de revisión, estos revisores deben comprobar:

● Verificabilidad: deberíamos hacernos la siguiente pregunta para decidir si un requisito es verificable. ¿puede probarse el requisito de manera realista? ● Comprensibilidad: las personas que vayan a utilizar el sistema o usuarios finales entenderán los requisitos de una manera correcta. ● Rastreabilidad: Evaluar el impacto que tiene el cambio en el resto del sistema, por lo tanto de ahí su gran importancia. ● Adaptabilidad: Se pueden cambiar los requisitos sin que repercutan a gran escala a otros requisitos del sistema.

Tabla 69. Comprobar mediante estos atributos la verificación de cada requisito por parte del equipo de revisión

4.1.4.6.2.2 Arquitectura de revisión de diseño (ADR) o Revisión de diseño preliminar (PDR)

El ADR se lleva a cabo para evaluar la adecuación técnica del diseño preliminar (también conocido como el diseño de nivel superior) del software como se muestra en la descripción del diseño de software preliminar.

Se realiza para evaluar la corrección técnica del diseño preliminar, es decir, alto nivel del software.

- Diagramas de flujo de datos del producto. - Descripción conceptual de estructuras y base de datos. - Nombres, unidades, y otros atributos de los elementos de datos. - Nombre y descripción funcional de cada módulo. - Especificación de interfaces para cada módulo. - Estructura de interconexión entre módulos. - Interconexiones entre módulos y estructura de datos. - Restricciones de tiempo. - Condiciones de excepción.

Tabla 70.Contenido del diseño arquitectónico

Los modelos arquitectónicos pueden incluir:

Modelo	Descripción
Modelo estructural estático:	Muestra los sistemas o componentes que han sido desarrollados como unidades separadas.
Modelo de proceso dinámico	Muestra cómo se organiza el sistema en tiempo de ejecución. Este modelo es diferente al modelo estático.
Modelo de distribución	Muestra el modelo de cómo se distribuyen los subsistemas entre los ordenadores.
Modelos de relaciones:	Muestra las relaciones, tales como el flujo de dato entre los subsistemas

Tabla 71.Modelos arquitectónicos

la Arquitectura de revisión del diseño (ADR) puede tener dos probables salidas: la ADR puede exponer problemas suficientes que se planea, una ADR subsecuente o la ADR puede exponer solo errores menores que pueden corregirse y revisarse en revisión de diseño crítico(CDR). La revisión de diseño crítico se desarrolla con el objetivo de aceptar o no el diseño detallado, comprobando que se ajuste a la especificación de requisitos software.

NOMBRE DEL PROCEDIMIENTO PARTE DE: (nombre y número del subsistema) LLAMADO POR: PROPÓSITO DISEÑADOR/FECHA(s)
PARÁMETROS: (nombres, modos, atributos, propósito) ASEVERACIÓN DE ENTRADA: (precondiciones) ASEVERACIÓN DE SALIDA: (poscondiciones) GLOBALES: (nombres, modos, atributos, propósitos, compartidos con) EFECTOS COLATERALES:

Tabla 72.Formato de patrón. Adaptado de [Richard E. Fairley]

4.1.4.6.2.3Revisión del diseño detallado

El DDR se lleva a cabo para determinar la aceptabilidad de los programas detallados diseños como se muestra en la descripción del diseño de software detallado para cumplir los requisitos de la SRD.

Son revisiones formales producidas durante el desarrollo del producto para asegurar que los requisitos, el concepto, el producto o el proceso satisfacen los requisitos de la etapa de desarrollo, los riesgos etc. Las revisiones de diseño típicas incluyen: revisión de requisitos, revisión de diseño de concepto/arquitectónico, revisión de diseño final.

- Descripción física de estructuras y bases de datos. - Especificación de diccionario para todos los elementos de datos. - Algoritmos detallados para cada módulo que se generará. - Adaptaciones necesarias para el código existente que será reutilizado. - Técnicas específicas de programación necesarias para resolver problemas especiales. - Procedimientos de inicio. - Pruebas para autorizaciones y manejo de excepciones. - Empacado de módulos en la instrumentación física.

Tabla 73.Contenido diseño detallado

El nivel apropiado de detalle de un producto para una PDR incluye la información contenida en:

Después del diseño detallado, se realiza una revisión crítica; la revisión requiere autorización del jefe del proyecto. La CDR se tiene al final del diseño detallado y antes de la instrumentación. La CDR proporciona un punto de decisión final para construir o cancelar el sistema. La CDR es en esencia una repetición de la PDR pero con el beneficio de un esfuerzo adicional de diseño.

Un proceso de revisión es:

"Un proceso o una reunión durante la cual un producto de trabajo, o un conjunto de productos de trabajo, se presenta al personal de proyectos, gestores, usuarios, clientes u otras partes interesadas para recabar observaciones o aprobación."

Las revisiones adquieren una gran importancia en el proceso de SQA a medida que estos documentos son productos en las fases iniciales del proyecto porque proporcionan la detección precoz y ayudan a prevenir errores de diseño y análisis en las etapas donde la detección y corrección son complicadas y a la vez costosas.

El jefe de revisión

El jefe de revisión es un factor esencial que afecta a la revisión de diseño detallado para lo cual debe cumplir con las siguientes características:

- Conocimiento y experiencia en revisión de proyectos. Buena relación con el jefe de proyecto y su equipo.
- Antigüedad en un nivel similar, si no más alto que el del líder del proyecto.

El equipo de revisión

Tener en cuenta el tamaño del equipo de revisión. Un equipo de revisión de tres a cinco se puede catalogar como un equipo eficiente, debido a la experiencia y enfoque entre los participantes. Un equipo grande tiende a crear problemas de coordinación, pérdida de tiempo revisión de sesión y disminuye el nivel de preparación.

Preparación para la revisión de diseño

Debe ser realizada por los tres miembros principales: el jefe de revisión, el equipo, y el equipo de desarrollo; para ello cada uno de los miembros debe centrarse en los diferentes aspectos de proceso.

Sus tareas principales como jefe de revisión son:

- Nombrar a los integrantes del equipo.
- Programar secciones de revisión.
- Distribuir el documento de diseño entre los integrantes del equipo.

Preparación de revisión del equipo

Los integrantes del equipo prevén revisar el documento de diseño y la lista de sus observaciones antes de la reunión de revisión. Una herramienta importante para garantizar la integridad de la revisión es la lista de verificación. La lista de verificación contribuye a la eficacia de la revisión del diseño.

El DDR reunión

La clave para una reunión exitosa en el DDR es la experiencia por parte del jefe de revisión; son las deliberaciones y pegas que se encuentran a la orden del día. Un programa de DDR incluye:

- Breve presentación del documento de diseño.
- Comentarios realizados por los integrantes del equipo de revisión.
- Verificación y validación en que cada uno de los comentarios se analiza para las acciones necesarias que los integrantes del proyecto tienen que realizar.
- Decisiones sobre el diseño del producto para lo cual determina el progreso del proyecto, en el cual se pueden tomar tres formas de decisión.

- Aprobación Completa: Permite el seguimiento a la siguiente fase del proyecto o en ocasiones puede estar acompañada por pequeñas correcciones a realizar por los integrantes del proyecto.

- Aprobación parcial: Permite el seguimiento a la siguiente fase en algunas partes del proyecto de los componentes de acción como pueden ser: correcciones, cambios, etc. que son indispensables para el resto del proyecto. A las partes de la fase restante se permite solo después de la finalización satisfactoria de los componentes de acción.

- Denegación de la aprobación: Exige realizar de nuevo DDR. Esta decisión se aplica en los casos de varios defectos especialmente los defectos críticos.

Revisión posterior a las actividades

- Preparar el informe de revisión, que resume los temas tratados y los puntos de acción.
- Establecer procedimientos de aseguramiento para garantizar el cumplimiento exitoso de todas las correcciones incluidas en la lista de componentes de acción.

4.1.4.6.2.4Revisión del plan de verificación y validación del software (V&V)

La revisión del plan de verificación y validación del software se lleva a cabo para evaluar la adecuación y exhaustividad de los métodos de verificación y validación definidos en los planes de verificación y validación.

Las principales técnicas de verificación son las revisiones y auditorías de software. Las revisiones de software se consideran el principal método de verificación, también sirven para validar el software ya que comprueban que lo que se está haciendo se ajusta a las necesidades de los usuarios. Aunque podemos observar otras técnicas complementarias.

Técnica/herramienta	Objetivo
Análisis de algoritmos	Verifica la funcionalidad de los algoritmos y recoge estadísticas sobre el consumo de recurso en tiempo de ejecución.
Análisis de simulación	Proporciona una evaluación del rendimiento y la información necesaria para planificar la capacidad de un sistema durante su diseño.
Auditores de código	Revisan el código fuente y determinan automáticamente si se siguen los estándares y practica de programación descritas previamente.
Generadores de referencia cruzadas	Producen listas de nombres de variables, procedimientos etiquetas, etc. determinando su ubicación dentro de un programa.
Analizadores de flujo de control	Determinan la presencia o ausencia de errores de flujo de control, es decir, secuencia incorrectas en la ejecución de un programa.
Analizadores y estimadores de tiempo de ejecución	Proporcionan información sobre la ejecución de un programa, es decir, tiempo de ejecución, consumo de CPU, etc.
Comprobación de interfaces	Analizar la inconsistencia y los flujos de información y control entre los módulos de un sistema.
Análisis de requisitos	Buscar errores sintácticos, inconsistencias lógicas o ambigüedades entre las entradas del sistema, salidas y procesos de datos
Análisis de trazabilidad de requisitos	Verificar que cada requisito del sistema está incluido en algún elemento del software. Garantizar que las pruebas se desarrollen sobre el software permitiendo verificar que se satisfacen los requisitos
Monitores de software	Supervisar la ejecución de un programa para ubicar áreas inadecuadas. Al terminar la ejecución el monitor crea informes que detallan el uso de los recursos del programa

Tabla 74.Técnicas de verificación y validación. Adaptado de [Mario G. Piattini]

Se realiza para comprobar que los métodos de verificación y validación definidos en el plan de verificación del software (PVVS) son los correctos.

- Detectar y corregir los defectos en la medida de lo posible durante el ciclo de vida del software.
- Disminuir los riesgos, desviaciones sobre presupuestos, calendario, programas de tiempo del proyecto.
- Mejorar la calidad y fiabilidad del software.
- Mejorar la visibilidad de la gestión del proceso de desarrollo.
- Valorar los cambios propuestos, así como sus consecuencias.

Tabla 75.Objetivos de la verificación y validación del software

Las revisiones de verificación y validación del software pueden ser:

Revisiones formales: Se utilizan dos técnicas:

- Inspección del software: Define el proceso, funciones, documentos, participantes, código fuente, verificación formal etc.

  Su objetivo es detección temprana de no conformidades y consiste en:

  1. La verificación que el producto cumple con los atributos de calidad.
  2. Detecta desviaciones de estándares y especificaciones
  3. Recopilación de datos.
  4. No revisa aspecto de estilo.
  5. Verificación de trazabilidad con respecto a las especificaciones de cada ítem.

- Las pruebas del software: Es una técnica dinámica de la verificación y validación ya que se lleva a cabo en una representación ejecutable del sistema. Comprende la implementación del software con los datos de prueba y revisa la salida y su comportamiento para confirmar que se desempeñe conforme a lo esperado.

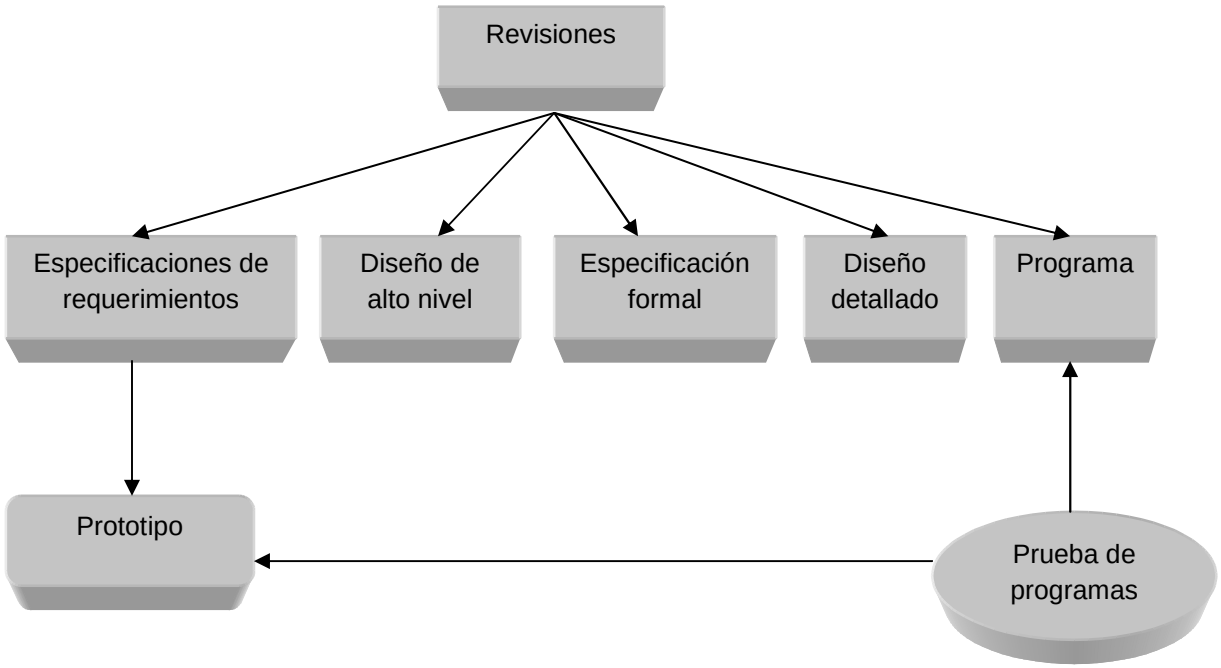


Figura 22.Verificación y validación estática y dinámica. Adaptado de [Ian Sommerville]

Las inspecciones del software se pueden utilizar en todas las etapas del proceso del software, las pruebas sólo se pueden usar cuando hay un prototipo o programa ejecutable.

4.1.4.6.2.5Auditoría funcional (AFU)

Esta auditoría se lleva a cabo antes de la entrega de software para verificar que todos los requisitos especificados en el DRS se han cumplido

Es una revisión que se desarrolla sobre el software justo antes de su entrega para verificar que cumple con todos los requisitos que se han definido en ERS. Antes de su entrega el jefe de proyecto es el responsable de su verificación. El jefe del proyecto debe obtener consentimiento previo del cliente por los medios adecuados e incluir un archivo con el nombre "léeme" o una carta de presentación con la cita de este archivo.

4.1.4.6.2.6Auditoría física (AFI)

Esta auditoría se lleva a cabo para verificar la consistencia interna del software y su documentación, y su disposición para su liberación.

Es una revisión que se realiza para verificar que el software y la documentación como el código fuente y todos los documentos asociados estén completos, sean consistentes tanto internamente uno del otro y que estén listos para enviarse. Antes de cada entrega el jefe de aseguramiento de la calidad es el responsable de verificar que el software físico y su documentación designados para la entrega, de hecho se entrega.

4.1.4.6.2.7Auditoría de procesos

Auditorías durante el proceso de las muestras del diseño se llevan a cabo para Verificar la consistencia del diseño incluyendo: código, especificaciones de la interfaz, diseño de implementación, requisitos funcionales con las descripciones de prueba.

Se desarrolla para verificar la consistencia del diseño e incluye lo siguiente durante el análisis.

- Código versus documentación de diseño.
- Especificaciones de las interfaces (software y hardware).
- Implementaciones de diseño versus requisitos funcionales.
- Requisitos funcionales versus descripciones de prueba.

Tabla 76.Análisis durante la auditoría de procesos

El personal del proyecto debe esperar auditorías aleatorias en el trabajo que desempeñe. Esta auditoría aleatoria consiste en visitas al área de trabajo de los equipos designados. Debe enviarse con un día de antelación para las visitas. Cuando la organización migra al nivel 5 de CMM, todo el trabajo se pondrá a disposición de todo el personal del equipo y de los auditores. El trabajo se desempeñará de una manera clara y estándar para que las auditorías sean posibles sin aviso previo.

El objetivo de la auditoría de calidad es proporcionar una evaluación objetiva e independiente de los procesos y productos para verificar estándares, especificaciones y procedimientos a los objetivos del proyecto. La auditoría puede ser sistemática (planificada) o excepcional (en función de los resultados obtenidos).

4.1.4.6.2.8Revisiones de gestión

La revisión de gestión se lleva a cabo periódicamente para evaluar la ejecución de todas las acciones y los elementos identificados en el PACS.

Revisión de la gestión es el nombre dado a la reunión periódica convocada para que directivos puedan obtener una visión general acerca de los problemas de la organización de la calidad del software. El informe de revisión de la gestión es elaborado por la unidad de SQA.

- Informes de rendimiento periódico, incluyendo las métricas de calidad. - Comentarios de satisfacción de los clientes. - Evaluación de los éxitos y fracasos en lograr los objetivos de la calidad, manteniéndose en el margen dentro del presupuesto, etc. - Informes de seguimiento de SQA programa de actividades y proyectos. - Resumen de eventos de calidad relacionados con los clientes, proveedores, subcontratistas, etc. - Revisión de los resultados sobre: estudios especiales, auditoría de calidad tanto interna como externa. - Identificación de nuevos riesgos de calidad del software y solución de los riesgos existentes. - Recomendaciones para las mejoras a introducir en el sistema de software de gestión de calidad (por ejemplo, el desarrollo de nuevos elementos de SQA, la compra de herramientas, etc.) que se presentará para su aprobación final

Tabla 77. Elementos de los informes de revisión de la gestión

Los principales objetivos de revisiones de gestión para evaluar el cumplimiento del sistema de SQA con la política de calidad de la organización:

- Valorar los logros de los objetivos de calidad establecidos para el sistema de la organización de gestión de la calidad del software.
- Poner en marcha las actualizaciones, objetivos y mejoras del sistema SQA
- Asignar recursos para las actividades de la calidad del software cuando sea necesario.

El sistema de gestión de calidad es una manera de alcanzar la calidad, su propósito es permitir a la organización satisfacer a los clientes, y para ello los recursos serán necesarios para su desarrollo, implementación mejora y mantenimiento, en la revisión de la gestión de calidad utilizaremos tres conceptos que definen el propósito de la revisión de la gestión:

Nombre	Descripción
Conveniencia del sistema	Se espera respuesta para lo que ha sido diseñado el sistema, en el cual satisface: estándares, requisitos y procedimientos
Adecuación del sistema	Se requiere por parte de la organización el mantenimiento de su actuación actual.
Eficacia del sistema	Es la respuesta de la organización de cumplir con las necesidades que la sociedad nos exige, para ello el sistema generará un usuario satisfecho, respondiendo a sus necesidades y cambios, si esto no se produce el sistema no es efectivo.

Tabla 78.Revisión de la gestión

Para ello la organización tiene la necesidad de revisar los datos que genera el sistema y determinar la respuesta dada, por tanto la revisión de gestión es completa, es la razón que la organización utiliza para alcanzar sus fines.

Los objetivos varían y para ello es recomendable planificar revisiones relativas a las características que queremos medir. Como resultado la organización puede obtener los siguientes tipos de revisiones:

- Inspección de diseño o programas: Detectar errores en los requisitos, el diseño, código etc. La revisión proporciona la verificación de errores.
- Revisiones del progreso: Obtener información del progreso del proyecto útil para la gestión, se refiere a costes, duración, planificación.
- Revisiones de la calidad: Requieren un estudio de los componentes del producto o documentación para encontrar diferencias entre código y documentación, diseño del componente y especificación etc. Y asegurarse que se cumplen los estándares de calidad definidos.

La revisión debe incluir la evaluación de oportunidades de mejora y la necesidad de efectuar cambios en el sistema de gestión de la calidad, incluyendo la política de la calidad y los objetivos de la calidad ISO 9000. El sistema de gestión de calidad es el medio en el cual la organización cumple con los objetivos, por lo tanto la revisión evalúa la necesidad de cambios en los objetivos y el proceso para el cual fue diseñado.

4.1.4.6.2.9 *Configuración del software de gestión de plan de revisión (SCMPR)*

El SCMPR se lleva a cabo para evaluar la adecuación y exhaustividad de los métodos de gestión de la configuración definida en el SCMP.

Se analiza la adecuación de los métodos recogidos en el plan de gestión de configuración del software. El jefe del aseguramiento de la calidad del software debe registrar el estado de la gestión de configuración cada mes de manera

independiente de los procedimientos del plan de gestión de configuración del software.

En el plan de gestión de configuración define una serie de tareas, tienen objetivos identificables:

1. Identificación de los elementos de gestión de configuración.
2. Gestionar los cambios.
3. Facilitar diferentes versiones en una aplicación.
4. Garantizar calidad en el software conforme evoluciona a lo largo del tiempo.

Estos objetivos mencionados anteriormente tienen por finalidad su desarrollado y se puedan reforzar en la fase que da soporte en el ciclo de vida del software. Por lo tanto algunas de sus actividades como la identificación, el control de la versión y el control de cambio ayudan al desarrollador del software a mantener el orden, si no fuese así habría una inestabilidad.

"El arte del progreso es preservar el orden entre el cambio, preservar el cambio entre el orden". Alfred North Whitehead.

4.1.4.6.2.10 Post-examen de la aplicación

Esta revisión se lleva a cabo a la conclusión del proyecto para evaluar las actividades de desarrollo en ese proyecto y formular recomendaciones para las acciones apropiadas.

Es la conclusión del proyecto, se evalúan las actividades de desarrollo y se envían recomendaciones de acciones para el futuro. El equipo del proyecto debe desempeñar un post-examen en todas las etapas, con el fin de proporcionar datos para proyectos futuros. Esto incluye revisiones de las etapas del proyecto en cuanto culminan y revisiones de los procesos de aseguramiento de la calidad. El equipo del aseguramiento de la calidad o el jefe de SQA, debe facilitar un informe de mejoras en el proceso de cada etapa y para el proceso de aseguramiento de la calidad mismo, al administrador.

Revisiones	Revisión	Productos revisados
Factibilidad del producto (PFR)	Se realiza una factibilidad del producto que determinará la continuidad del proyecto, como resultado de la revisión se puede continuar o redirigir a través de la definición del sistema y plan del proyecto	- Definición del sistema. - Plan de proyecto
Especificación de requisitos del software (SRR)	Se realiza una revisión de requisitos del software para determinar la consistencia entre el plan de proyecto, definición del sistema, requisitos de producción, plan	- Especificación de los requisitos para la producción del software. - Documento de usuario

	de verificación, documento de usuario	- Plan de verificación preliminar
- Diseño arquitectónico. - Diseño detallado	- Evaluar consistencia respecto a la especificación de producción del software. - Revisión crítica cuyo objetivo es determinar si se acepta la especificación del diseño del software	- Diseño estructural - Diseño detallado
- Verificación - validación	- Se prepara para describir los resultados de todas las revisiones, auditorías y pruebas durante el ciclo de desarrollo. - Evalúa el software al final del ciclo de revisión para determinar su conformidad con los requisitos	- Plan de verificación del software - Recorridos e inspecciones del código fuente - Plan de prueba de aceptación
Documentación para el usuario	Proporciona un mecanismo entre el cliente y equipo de desarrollo. El manual de usuario forma parte de la especificación de requisitos	- Manual de usuario
Administración configuración software	- Revisión técnica formal: Planifica, controla y atiende las necesidades que se presentan - Auditoría de la configuración: completa la revisión formal y los elementos de configuración son correctos y la documentación es consistente	- Plan de la gestión de configuración

Tabla 79. Revisiones y logros durante las fases del ciclo de vida

4.1.4.6.3 Otras revisiones y auditorías

Otros comentarios y las auditorías pueden incluir la revisión de la documentación de usuario (UDR). Esta revisión se lleva a cabo para evaluar la adecuación (por ejemplo, integridad, claridad, exactitud y facilidad de uso) de la documentación del usuario.

Revisiones por homólogos

Los métodos de revisión por pares son: Inspecciones y recorridos, se pueden utilizar para examinar de forma metódica los productos de trabajo a través del ciclo de vida del sistema como son:

- Los requisitos.
- Especificaciones de diseño.
- Plan de prueba.
- Código fuente.
- Manual de usuario.
- Entre otros.

Hoy en día, con la aparición de herramientas de diseño computarizado, incluyendo las herramientas CASE, por un lado, y los sistemas de paquetes de software por otra parte, algunos profesionales tienden a disminuir el valor de las revisiones manuales, tales como las inspecciones y recorridos.

Sin embargo, estudios anteriores de software, así como los últimos resultados de la investigación empírica ofrecen mucha evidencia convincente de que las revisiones de pares son altamente eficientes.

Lo que diferencia a un recorrido de una inspección es el nivel de formalidad. La inspección destaca el objetivo de la acción correctiva. Considerando que las conclusiones de un recorrido se limitan a comentarios sobre el documento revisado.

Las inspecciones, en lugar de recorridos, consideran por lo tanto contribuir más significativamente con el nivel general de SQA.

- Elaboración de listas de control de inspección desarrolladas para cada tipo de documento de diseño, así como la codificación del lenguaje y la herramienta. - Análisis periódico de la eficacia en las inspecciones anteriores para mejora de las metodologías en las distintas inspecciones. - La formación de profesionales competentes en temas de proceso de inspección. Los empleados formados pueden servir como una reserva de inspectores profesionales disponibles para futuros proyectos - En el plan de actividades del proyecto requiere la inspección de programas y asignación de recursos para corregir defectos.

Tabla 80.La inspección se basa en una infraestructura completa

Las organizaciones suelen modificar estos métodos de inspección y recorrido adaptándolas al personal de desarrollo, las unidades de SQA, productos del software desarrollado, la estructura etc.

Personal para las revisiones externas

El equipo de revisión por pares es de tres a cinco participantes. En algunos casos de uno a tres participantes. Un equipo de revisión por pares recomendado incluye:

- Un líder de revisión.
- El autor.
- Profesionales especializados.

El líder de revisión

- El papel del líder de revisión es de moderador en las inspecciones, y coordinador en los recorridos.
- Conocer el desarrollo de proyectos en la actualidad y adaptarse a las tecnologías.
- Sostener buenas relaciones con el equipo de desarrollo y autor.
- Experiencia en la coordinación y dirección de reuniones.

El autor

El autor es un participante ya sea de inspección o de recorrido.

Profesionales especializados

Los profesionales especializados difieren en la revisión en la cual participan.

Inspecciones	- Diseñador: Encargado del análisis y sistema de la revisión del software. - Codificador: Contribuye a la detención de defectos y pueden inducir a errores de codificación. - Probador: Identificación de errores de diseño que se detecta en la fase de prueba.
Recorrido	- Ejecutor de estándares: Localiza las desviaciones de las normas y procedimientos. Lo cual afecta a largo plazo a la efectividad del equipo los errores de este tipo, reduce la eficacia del equipo que se va a mantener en el sistema. - Experto en mantenimiento: La documentación cuya integridad y precisión es importante para la actividad de mantenimiento. - Detecta errores que pueden impedir corrección de errores. - Representante de los usuarios: Participación de un interno (cliente que participa en la organización) o usuario externo el equipo de recorrido revisa el sistema desde el punto de vista usuario-consumidor en lugar de diseño-proveedor.

Tabla 81.Profesionales especializados que participan en los métodos de revisiones de pares

Revisión de la documentación

La documentación producida en el final de una revisión de inspección es mucho más completa que la de una revisión de recorrido.

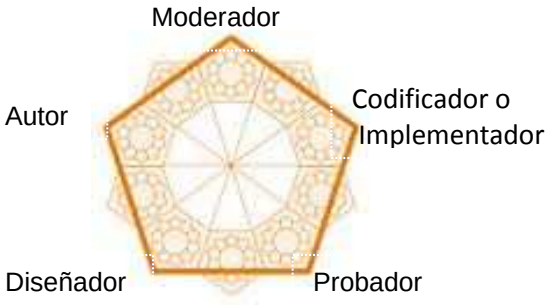
Los dos documentos se producirán después de una revisión de inspección y mediante los siguientes cometidos:

1. El objetivo primordial es gestionar una documentación completa de errores detectados para su seguimiento y corrección.

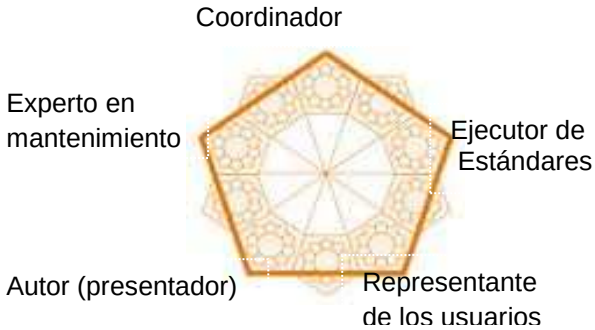
2. Informe de resumen debido al período de revisiones de Inspección. El informe sirve principalmente como entrada para el análisis encaminado a mejorar el proceso de inspección y las medidas correctivas que van más allá del documento o proyecto específico.

Participantes

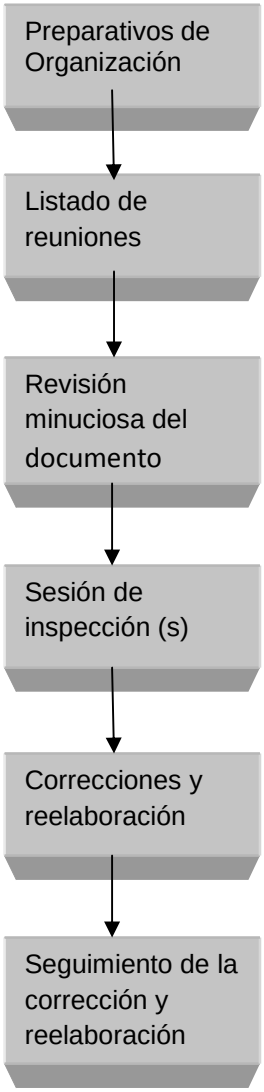
Inspección



Recorrido



Procesos



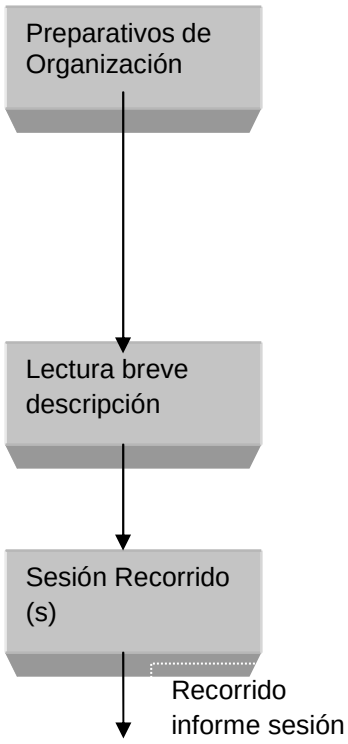


Figura 23.Inspección vs recorrido – participantes y procesos

4.1.4.7 Prueba

En esta sección se deberán identificar todas las pruebas no incluidas en la verificación de software y un plan de validación para el software cubierto por el PACS y se indicarán los métodos a utilizar.

Su principal objetivo es la realización de un conjunto de actividades que a la vez se descomponen en distintas tareas para el buen desenvolvimiento del software.

- Actividad diseño de planes de prueba.
- Actividad ejecución de casos de prueba.
- Actividad revisión de planes de prueba.
- Actividad diseño de planes de prueba.

Cabe resaltar la necesidad de que los casos de prueba estén identificados con el fin de facilitar las referencias futuras al ser ejecutadas, así mismo los datos de entrada como de salida esperada, el plan de pruebas no es un fin en sí mismo, sino un medio para desarrollar ordenadamente los procesos de prueba.

Tareas

Descripción del plan de pruebas

Plan de Pruebas
1. Identificación del documento y el producto a probar. 2. Objetivos del plan. 3. Planificación temporal y de recursos. 4. Entornos de prueba. 5. Elementos auxiliares. 6. Prueba X.

Tabla 82.Plantilla pan de pruebas. Adaptado de [tecnicas cuantitativas para la gestión de Ing software]

Tarea: Descripción plan de pruebas
Objetivo: Desarrollar un plan de pruebas, dichas pruebas se irán desarrollando a lo largo de las fases del proyecto, en el cual se seguirán las políticas mencionadas en dicho plan
Descripción: la creación de plan de pruebas debe estar relacionado con la plantilla del plan de pruebas, hay que tener en cuenta que se trata de un documento vivo que puede haber variaciones a lo largo del proyecto de pruebas
Entrada: - Caso de uso de elementos - Modelo de clases - Controles de caso de uso
Producto generado: - Plan de pruebas: se debe especificar en cada plan de prueba la siguiente información: • Información detallada: ➢ Fecha de creación; <*dd-mm-yyyy*>

➢ Fecha de revisión: <*dd-mm-yyyy*> ➢ Identificador del producto: <identificador> ➢ Producto: <nombre del producto> <país> <versión> ➢ Modificado Por: <nombre, apellidos> ➢ Tiempo de ejecución: <*h-m-s*> • Descripción: Se debe describir el producto que se ha generado, mencionando características, se bebe incluir una lista correspondiente al plan con los puntos más destacados incluidos en las pruebas. • Enfoque que se da al caso de prueba. • Caso de uso con los que se relaciona. • Pruebas automáticas.
Salida : - Información detallada: ➢ Fecha de creación; <*dd-mm-yyyy*> ➢ Fecha de revisión: <*dd-mm-yyyy*> ➢ Identificador del producto: <identificador> ➢ Producto: <nombre del producto> <país> <versión> ➢ Modificado Por: <nombre, apellidos> ➢ Tiempo de ejecución: <*h-m-s*> - Descripción. - Casos de uso en el que se relaciona. - Pruebas automáticas.
Rol: Analista de Prueba
Observaciones:

Tabla 83.Descripción plan de pruebas. Adaptado de [técnicas cuantitativas para la gestión de Ing software]

Tarea: Definición de casos de prueba

Definición de casos de prueba

Caso de prueba 1. Elemento a probar 2. Objetivo del caso 3. Entorno de pruebas a usar y condiciones a reproducirse 4. Modo de ejecución 5. Datos de entrada 6. Salida esperada

Tabla 84.Casos de prueba. Adaptado de [técnicas cuantitativas para la gestión de Ing software]

Tarea: Definición de casos de prueba
Objetivo: Nos permite probar todas las funcionalidades, por lo tanto es de gran importancia tener un buen criterio a la hora de desarrollarlos, el empleo de casos de usos no puede dar una gran cantidad de variables.
Descripción: desarrollo caso de prueba
Entrada: - Controles de caso de uso - Usos - Casos - Casos de usos - Detalles

Producto generado: - Caso de uso: Se debe especificar en cada caso de prueba la siguiente información: • Identificador del caso de prueba: <id/nombre/Sistema/proyecto> • ID caso de uso: – Tipo de prueba: ➢ Prueba funcionalidad ➢ Prueba de desempeño ➢ Prueba de interfaz de usuario • Nombre: <nombre caso de uso> • Datos de inicio: Estado de inicio en que se encuentra la aplicación. • Persona Responsable: Persona que ejecuta el caso de prueba. • Tipo de ejecución: ya sea manual y/o automática. • Ejecución del caso de prueba: ➢ Elemento a probar. ➢ Condición. ➢ Valores. ➢ Resultado esperado. ➢ Resultado obtenido.
Salida: - Identificador caso de prueba - Tipo de prueba - Nombre - Persona Responsable - Tipo de ejecución - Ejecución del caso de prueba
Rol: Analista de Prueba
Observaciones :

Tabla 85.Definición caso de pruebas. Adaptado de [técnicas cuantitativas para la gestión de Ing software]

Tarea: Descripción de scripts de pruebas

Tarea: Descripción de scripts de pruebas
Objetivo: Obtener los scripts automatizables para los casos de pruebas.
Descripción: Crear los scripts correspondientes para llevar a cabo los casos de prueba que se hayan establecido como automatizables en el documento de caso de prueba.
Entrada: - casos de prueba - controles de caso de uso
Producto generado: Script caso de prueba - permite le ejecución en cualquier lenguaje siempre y cuando se establezca una relación con la aplicación.
Rol: Analista de Prueba
Observaciones:

Tabla 86.Descripción de scripts de prueba. Adaptado de [técnicas cuantitativas para la gestión de Ing software]

2. Actividad: Ejecución de casos de prueba

La ejecución de casos de prueba se realiza mediante una ejecución manual y la creación de los scripts automáticos para cada uno de los casos de prueba, es necesaria la creación de un repositorio de todos los script que se han desarrollado para los casos de prueba a fin de mantener una estructura de almacenamiento para su reutilización.

Tarea: Preparación para la ejecución de casos de prueba
Objetivo: Preparar la ejecución de los casos de prueba, registrando los resultados en la herramienta de gestión de pruebas y gestión de incidencias
Descripción: Instalar un entorno que simule lo que podría ser la aplicación
Entrada: - Software - Script casos de prueba - Casos de prueba - Aplicación para ser instalable
Producto generado: - Entorno para las pruebas: Documento que contendrá todas las especificaciones referentes al sistema contendrá todas las pruebas tanto de hardware como de software.
Salida: - Entorno para las pruebas ➢ Especificaciones hardware: • Procesador • Velocidad • Memoria(Ram) ➢ Especificaciones software: • Tipo de sistema operativo • Software versión
Rol: - Probador
Observaciones :

Tabla 87.Preparación para la ejecución de casos de prueba. Adaptado de [técnicas cuantitativas para la gestión de Ing software]

Tarea: Puesta en Marcha de los casos de prueba
Objetivo: - Ejecutar los casos de prueba - Desarrollo de gestión de incidencias - Obtener los resultados de los casos de prueba
Descripción: Ejecución de la aplicación. La aplicación debe encontrarse en estado de inicio para realizar los casos de prueba, si estas pruebas son automatizables se deberían desarrollar con los scripts oportunos, en caso de no serlo se debe documentar las especificaciones en el caso de pruebas. Obtención de los resultados obtenidos para verificar si son satisfactorios.
Entrada: - Casos de prueba - Scripts - Ejecutable
Producto generado: - Resultado obtenido de los casos de prueba Este elemento tiene que describir la siguiente información detallada

➢ Resultado obtenido ➢ Resultado esperado ➢ Identificador del elemento ➢ Resolución ➢ Conclusiones
Salida: - Identificador del elemento - Resultado obtenido - Resultado esperado - Resolución - Conclusiones
Rol: Probador
Observaciones:

Tabla 88.Puesta en marcha de los casos de prueba. Adaptado de [técnicas cuantitativas para la gestión de Ing software]

Actividad Revisión de planes de prueba

Tarea: Revisión de casos de prueba
Objetivo:
Descripción: las revisiones técnicas formales son el filtro más efectivo des de un punto de vista de aseguramiento de la calidad, es un medio eficaz para mejorar la calidad del software, verificar si el resultado obtenido es causa de: - Errores pasados por alto - Errores amplificados - Nuevos errores generados
Entrada: Obtención de los resultados de la ejecución de casos de prueba
Producto generado: - Obtención de los resultados de casos de prueba; para ello debe estar acompañado de la siguiente información: ➢ Conclusiones ➢ Resolución
Salida: - Conclusiones - Resolución: ☐Correcto ☐Incorrecto
Rol: - Probador
Observación:

Tabla 89.Revisión de casos de prueba

Tarea: Notificación de defectos al equipo de desarrollo
Objetivo: Dar a conocer al equipo de desarrollo las distintas anomalías, incidencias, resultados no esperados para tomar las medidas oportunas
Descripción: Entregar al equipo de desarrollo una lista de problemas de la revisión notificando los errores cumpliendo con los siguientes propósitos: 1. Identificación de cada una de las aéreas del problema en el producto 2. Lista de verificación para cada producto que tenga la probabilidad de ser revisado.
Entrada: Resultados obtenidos en los casos de prueba
Producto generado: Se entregará al equipo de desarrollo, ya que este tenía una visión del producto en lo referente a: - Precisión - Modularidad

Salida: - Plan de prueba - Caso de Prueba
Rol: - Jefe de equipo de desarrollo - Probador
Observación:

Tabla 90.Notificación de defectos al equipo de desarrollo. Adaptado de [técnicas cuantitativas para la gestión de Ing software]

4.1.4.8 Problema de informes y medidas correctoras

En esta sección se deberán describir prácticas y procedimientos y solución de problemas de los elementos, desarrollo del software y proceso de mantenimiento.

El informe de problemas se realizará a través de una plantilla que será una aplicación y esta aplicación se encargará de que todos los campos de la plantilla se llenen adecuadamente. A continuación explicaremos en que se basa el informe de problemas:

Los valores de severidad son los siguientes:

- Importante: Da como resultado un requisito no satisfecho.
- Trivial: No afecta la ejecución de la aplicación o mantenimiento.
- Menor: No es importante ni trivial.

El tipo de defectos de documentación incluye material poco claro, ambiguo, redundante, incompleto, contradictorio.

El tipo de defectos incluye sintaxis, lógica, permitir valores erróneos de variables, inseguridad.

El jefe del aseguramiento de la calidad y el equipo de éste crean y conservan una base de datos de informes de problemas en el cual detallan deficiencias, discrepancias, anomalías. Manifiestan que los defectos se registrarán de manera consistente y dan camino para solventar los defectos producidos. Los informes de problemas deben seguirse de acuerdo al plan de gestión de la configuración del software.

Cuando se descubre un problema, el administrador del aseguramiento de la calidad asigna el informe del problema al comité de control de cambios (CCC). El comité de control de cambios evalúa el informe y asigna prioridades. Después el CCC asigna el informe al equipo de desarrollo o al equipo de gestión de la calidad para su solución.

EL comité de control de cambios determina el tiempo empleado para la solución del informe con base a la prioridad y el informe de resultado del análisis. Cuando se corrige el informe del problema, el equipo de

aseguramiento de la calidad examina los resultados y el administrador del aseguramiento de la calidad informa al comité de control de cambio si el informe es aceptado o rechazo.

1. Defecto número: 2. Propone:
3.Documentos/secciones afectadas:
Código afectado*: 4.Paquete(s):_____ 5.Clase(s):_____ 6.Método(s):_____ 7. Severidad: _____ 8. Tipo:_____
9.Etapa inyectada**: Requisitos ☐ Archivo ☐ Código ☐
10.Descripción detallada: _____ 11.Solución: 12.Estado cerrado/abierto:___
Terminar: Descripción y plan inspeccionado:_____ 14.Codigo solución y plan de pruebas inspeccionado:_____ 15.Cambio aprobado para incorporación:_____
*: Defectos en código fuente. **: Primeras etapas con el defecto.

Tabla 91.Plantilla de un informe de problema: Adaptado de [Eric J. Braude]

Seguimiento de acciones correctivas

Se informará a la dirección del proyecto de los hallazgos obtenidos durante las verificaciones.

La aplicación de acciones correctivas ayuda a cumplir con los objetivos del sistema para el aseguramiento de la calidad, en cuanto a servicios de consulta de información, seguimiento de las acciones correctivas, control y auditorías.

Acción Correctora Número	Número Auditoría	Informe de confirmación Nº
Actividad Afectada/obra Afectada		
Comentario:		
Acción correctora:		
Acción preventiva:		
Documentación afectada:		
Realizado: Firma:	Aprobación Firma:	Fecha

Tabla 92.Informe de acción correctora

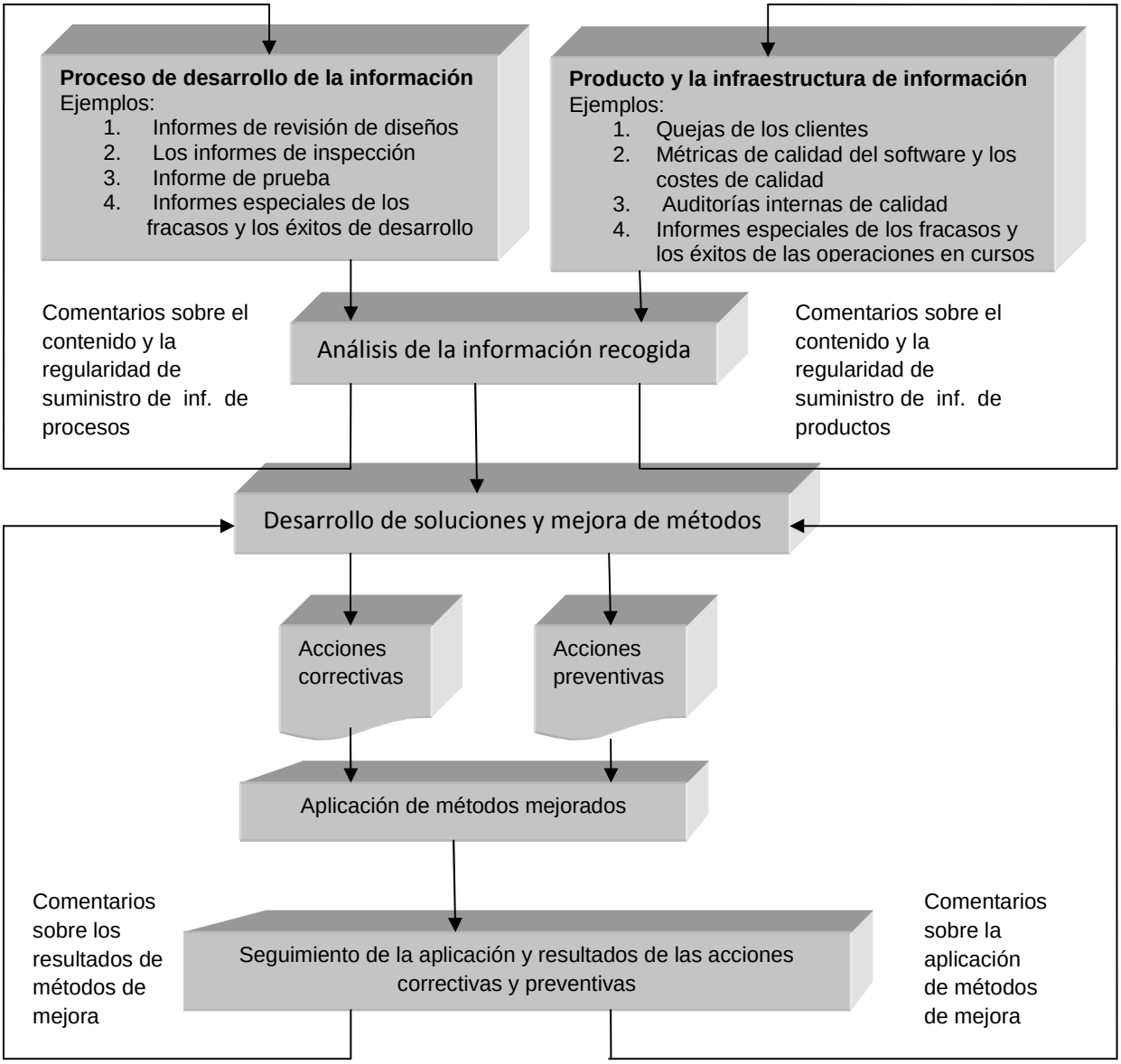


Figura 24.Acciones correctivas y preventivas de procesos. Adaptado de [Daniel Galin]

Las principales fuentes de información interna son:

1. Proceso de desarrollo de software.
2. Mantenimiento de software.
3. Infraestructura de SQA.

Procedimientos de software de gestión de calidad

Como fuente externa de información tenemos principalmente las estadísticas de los clientes de aplicaciones y quejas de los clientes

Fuente	Descripción
Proceso de desarrollo de software	- Informe software de gestión de riesgo. - Informes de revisión de diseños. - Informes de inspección. - informes Walkthrough. - Informes de opinión por parte de expertos. - Revisiones prueba. - Informes especiales de los fracasos y los éxitos de desarrollo. - Propuestas por los miembros del personal.
Mantenimiento de software	- aplicaciones estadísticas de los clientes. - software solicita el cambio iniciado por aplicaciones cliente. - software solicita el cambio iniciado por el personal de mantenimiento. - informes especiales sobre las fallas de mantenimiento y éxitos. - las propuestas sugeridas por los miembros del personal.
Infraestructura de SQA	- Informes internos de calidad de la auditoría. - Informes de auditoría externa de calidad. - Rendimiento de seguimiento de personal capacitado y certificado. - Las propuestas sugeridas por los miembros del personal.
Procedimientos de software de gestión de calidad	- Informes de avance del proyecto. - Métricas de calidad del software informes. - Software de informes de calidad de costos propuestas de los miembros del personal.
Fuentes externas de información	- Quejas de los clientes. - Estadísticas de servicio al cliente. - Propuestas sugeridas en el cliente.

Tabla 93.Información utilizada para las acciones correctivas y preventivas mediantes fuentes y documentación

4.1.4.9 Herramientas, técnicas y metodologías

En esta sección se deberán identificar las herramientas de software, técnicas y métodos utilizados para apoyar los procesos de SQA.

Las técnicas del aseguramiento de la calidad del software incluyen auditorías de estándares, rastreo de requisitos, verificación de diseño, inspecciones de software y verificación de los métodos formales. Estas herramientas del aseguramiento de la calidad del software consisten en programas de verificación, listas de verificación, métricas y sellos de aceptación.

- Listas de verificación que se utilizan en reuniones formales, para revisiones de documentos y para inspecciones.
- Las listas de verificación se usarán para revisar la calidad de las siguientes actividades y documentos: revisión de diseño preliminar, revisión de diseño crítico, revisiones de pruebas, auditoría de configuración funcional, especificación de requisitos software, documento del diseño de software, plan de administración del proyecto de software.
- Se usan listas de verificación y formas separadas para los aspectos de auditoría de software.

Tabla 94.Listas de verificación. Adaptado de [Eric J. Braude]

El objetivo de estas herramientas es hacer que el trabajo de los equipos de desarrollo y mantenimiento sea más eficiente y eficaz. Hoy en día hay muchas herramientas que se encuentran en el mercado y son eficaces y de fácil uso.

Nos centraremos en la importancia que tiene la herramienta CASE y para ello desarrollaremos con más detalle su importancia y uso. Las herramientas CASE sirven de fuente para facilitar la cantidad de esfuerzo en el desarrollo de sistemas de software cada vez más complejas y grandes.

Definición de herramienta CASE*: Se han informatizado las herramientas de desarrollo de software que apoyan el desarrollador cuando se realiza una o más fases del ciclo de vida del software y / o mantenimiento de software de apoyo. Daniel Galin*

- Las herramientas CASE para el desarrollo del software han sido un gran avance en la mejora de la productividad y la calidad del mantenimiento del software
- Reutilización del software mediante el apoyo de la documentación, actualización y normalización.
- Contribución a la calidad del software mediante la reducción de los errores.

Tabla 95.Puntos fundamentales de las herramientas Case

Las diferencias que presentan las herramientas CASE clásicas y herramientas CASE real

Las herramientas CASE clásicas	Las herramientas CASE real
Se han informatizado las herramientas de desarrollo de software que apoyan el desarrollo (como depuradores interactivos, los compiladores y sistemas de control de avance del proyecto) cuando se realizan una o más fases de ciclo de vida del software y mantenimiento del software de apoyo	Se distinguen dos clase de herramientas: - Herramientas CASE superior: Soportan las fases de análisis y diseño, - Herramientas CASE inferior: Soportan la fase de codificación. - El principal componente de las herramientas CASE real es el repositorio que almacena toda la información relacionada con el proyecto. La información del proyecto se acumula en el repositorio y se modifican los cambios en el inicio de las fases de desarrollo y fase de mantenimiento. El repositorio de la fase de desarrollo sirve como base para la siguiente fase. La información sobre la fase de desarrollo almacenado en el repositorio ofrece apoyo para la fase de mantenimiento en el que las tareas de corrección, de adaptación y mejora de la funcionalidad se llevan a cabo. Por lo tanto las herramientas CASE son capaces de producir la documentación del proyecto completo y modificada en cualquier momento, como ejemplo podemos citar: Herramientas integradas CASE, herramientas de referencia cruzadas sobre la base de repositorio de datos, herramientas de ingeniería inversa

Tabla 96.Diferencia entre las herramientas CASE clásicas y herramientas CASE real

Las herramientas CASE real de la calidad del producto nos aportan:

- Identificación de las desviaciones de las necesidades del diseño.
- Identificación de las incoherencias de diseño.
- La generación automática del código basado en los registros de diseño del repositorio, sin errores esperados.
- Correcciones y cambios de alta calidad realizados durante el desarrollo debido a herramientas que se encuentran en el repositorio.
- Generación automática del repositorio de los sistemas de herencia a las herramientas CASE de ingeniería inversa, permite el desarrollo eficiente de las nuevas generaciones de software con el sistema de garantía máxima de calidad del software.
- Conformidad con el diseño y la codificación de las instrucciones de documentación en el repositorio realizado por la automatización.

Tabla 97.Aportación de Las herramientas CASE real de la calidad del producto

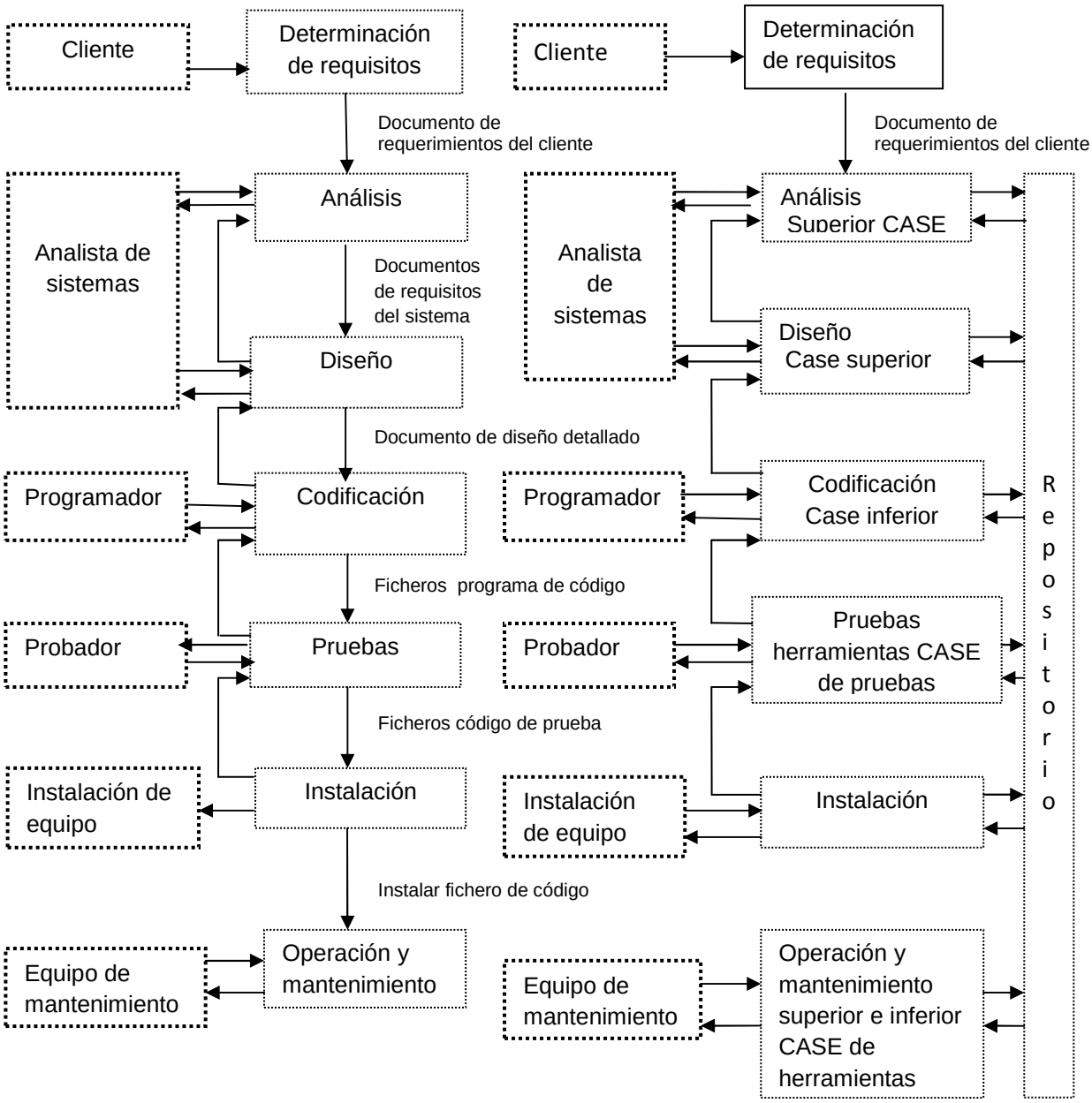
Las herramientas CASE real para la calidad del mantenimiento

Aplicación de herramientas CASE afecta a cuestiones de calidad para todos los componentes de servicio de mantenimiento:

- La codificación automática llevada a cabo por herramientas CASE elimina errores de codificación.
- Herramientas de referencia cruzadas en el repositorio facilitan la planificación y evitan errores en el diseño.
- Documentación completa y modificada proporcionada por el repositorio, asegura cambios y complementos en el sistema actual y permite la revisión de las aplicaciones del sistema existente.

Tipo de herramienta CASE	Apoyo prestado
Edición y diseño de diagramas	Edición de texto, diagramas y la generación de diagramas de acuerdo a los registros del repositorio.
Repositorio de consulta	Visualización de texto de diseño, gráficos etc. Consulta de referencias cruzadas y seguimiento.
Documentación automatizada	Generación automática de la documentación solicitada dependiendo de repositorio en los registros.
Diseño de apoyo	Diseño de imágenes grabadas por el analista de sistemas y la gestión del diccionario de datos.
Edición del código	Compilar, interpretar, aplicar el código de depuración interactiva para un lenguaje específico de decodificación o herramientas de desarrollo.
Gestión de configuración	Transformación de los registros de diseño en prototipo o aplicaciones software compatible con un lenguaje o herramientas de desarrollo.
Pruebas de software	Distintas clases de prueba que se han desarrollado con más detalle en el capítulo de pruebas de software.
La ingeniería inversa (re-ingeniería)	La construcción del repositorio de software y documentos de diseño, basado en el código. Una vez que el repositorio de software se encuentra disponible, se pueden actualizar y utilizar para la generación automática de nuevas versiones del sistema.
Gestión de proyectos y las métricas del software	Apoyar el control del avance de los proyectos de desarrollo de software de seguimiento de los horarios y el cálculo de métricas de la productividad y los defectos.
Generación de código	Transformación de los registros de diseño en prototipos o aplicaciones de software compatibles con un lenguaje de desarrollo de software dado (o herramientas de desarrollo).

Tabla 98.Herramienta CASE y el apoyo que brinda a los desarrolladores



a.) Desarrollo tradicional durante el ciclo de vida

b.) Verdaderas herramientas CASE Apoyados por desarrollo del ciclo de vida

Figura 25.Desarrollo tradicional durante el ciclo de vida vs desarrollo CASE apoyados por el ciclo de vida. Adaptado de [Daniel Galin]

En la siguiente tabla se enumeran las herramientas CASE que aportan a la calidad.

Causa de errores de software	Herramientas CASE clásicas	Herramientas CASE real
1. Requisitos definición errónea		**Prácticamente no contribuyen** Información informatizada de los requisitos de corrección rara vez es posible.
2. Cliente-desarrollador errores de		**Prácticamente no contribuyen**

comunicación		La identificación de los fallos automatizados en la comunicación es imposible. Ya que los fallos de comunicación pueden ser localizados cuando una información de cambio resulta incompatible con el repositorio de información.
3. Desviaciones intencionales de las necesidades del software		**Alta participación** La información almacenada en el repositorio, las desviaciones de requisitos registrados e identificados como inconsistentes y etiquetados como errores. Pueden ser basadas en herramientas de consulta, herramientas de seguimiento y referencias cruzadas.
4. Errores de diseño lógico		**Alta participación** (1) Re-ingeniería permite la generación automática del diseño de los sistemas de herencia y su registro en un repositorio.
5. Errores de diseño lógico		**Alta participación** (2) El uso del repositorio es para identificar omisiones de diseño, modificaciones y complementos incompatible con el repositorio de registros.
6. Errores de codificación	**Muy alta participación** Aplicación de los compiladores, intérpretes y depuradores interactivos.	**Muy alta participación** La codificación es automática, sin errores. La Aplicación de herramientas CASE es muy bajo, genera código automático que alcanza coherencia en el diseño registrado en el repositorio.
7. El incumplimiento de las instrucciones de codificación y documentación	**Limitada participación** El uso de editores de texto y la auditoría de código facilita la identificación de incumplimiento y admite la normalización de la estructura y el estilo de los textos y de código.	**Muy alta participación** La codificación es automática, sin errores. La Aplicación de herramientas CASE es muy baja, genera código automático asegura el cumplimiento de las instrucciones documentación y la codificación.
8. Deficiencias en el proceso de prueba	**Alta participación** Gestión automatizada de pruebas y correcciones por	**Alta participación** Evita los errores de codificación y reduce los errores de diseño las

	seguimientos reduce errores. Las herramientas automatizadas de pruebas de regresión completa y realizar pruebas de carga automática.	herramientas CASE integrado. La aplicación de instrumentos del repositorio en los cambios y correcciones durante el proceso de desarrollo previene la mayoría de los errores de software.
9. Errores de procedimiento	**Alta participación** Revisiones, control de versiones, instalación del software por medio de herramientas de software de gestión de la configuración.	**Limitada participación** El uso de modificaciones y documentación completa si el diseño ha sido revisado varias veces, se espera evitar que muchos de los errores de mantenimiento ocasionadas por documentación inexacta.
10. Documentación de los errores	**Limitada participación** Aplicación de los editores de texto.	**Alta participación** El uso del repositorio genera automáticamente la documentación completa y actualizada antes de cada corrección o cambio.

Tabla 99.Herramientas CASE y la calidad de los productos de software. Adaptado de [Daniel Galin]

Entornos asociados a la metodología

El establecimiento de metodologías de análisis y diseño en la ingeniería del software en los primeros años de existencia dio lugar a la aparición de diversas herramientas de soporte de dicha metodologías. Muchas de esas herramientas se han denominado herramientas CASE su principal ventaja es que suelen construir entornos de trabajo bien integrados y con una interfaz de usuario uniforme.

El repositorio CASE alberga todos los elementos de información contemplados en la metodología de soporte, podemos citar como ejemplos: metodología de análisis y diseño estructurado basado en el empleo de diagramas de flujos de datos, así como las características de cada dato y proceso.

En el entorno de análisis y diseño permite la realización separada de cada diagrama de flujo y proceso así como el diccionario de datos. La herramienta CASE facilita el mantenimiento de las diferentes descripciones o diagramas del producto del software. Todos estos conceptos que se exponen en este párrafo vienen con más detalle explicado en el capítulo de diseño del software.

4.1.4.10 Control de medios

En esta sección se deberán identificar los métodos e instalaciones que se utilizarán para identificar los medios de comunicación de cada producto, equipo de trabajo, entregar documentación requerida para almacenar los medios de comunicación y protección de los medios de comunicación.

Los métodos e instalaciones para conservar, guardar, confirmar y documentar las versiones de código terminadas durante el transcurso de las etapas del ciclo de vida del software se especifican en el plan de la gestión de configuración.

El equipo de gestión de la configuración del software verifica que se sigan los procedimientos de control de código. Las etiquetas de medios para el código base tiene marcas separadas para verificación, duplicación y validación.

Después de terminar una prueba formal validada o presenciada por personas de la gestión de configuración del software en la etiqueta de medios para el código base y la persona responsable del aseguramiento de la calidad firma la etiqueta.

El equipo de gestión de configuración verifica que el medio del software se construya y configure según el PGCS y que los cambios autorizados hayan sido instalados y probados.

Además el equipo de gestión de configuración del software se duplique utilizando solo procedimientos identificados en el plan de gestión de configuración del software. La aceptación del aseguramiento de la calidad se indica por un sello de la gestión de configuración del software en la etiqueta del medio. Los informes de auditoría son una evidencia más que se cumplieron los procedimientos del aseguramiento de la calidad.

4.1.4.11 Proveedor de control

En esta sección se hará constar las disposiciones para asegurar que el software proporcionado por los proveedores cumple con los requisitos establecidos.

Un proyecto de software se puede utilizar u obtener productos que se hayan comprado. La información que se obtenga por parte del proveedor debe estar disponible para monitorizar eficiencia y eficacia mediante el cumplimiento de los requisitos establecidos.

Para evaluar la eficacia y eficiencia en el aseguramiento de la calidad del software está el papel que desempeñan los proveedores, los proveedores simbolizan la entrada del proceso principal, por lo tanto debemos controlar la calidad de suministro, para poder garantizar la calidad del producto o servicio final.

Mantener una evaluación fija al proveedor brinda los siguientes beneficios entre otros:

- Progreso en la eficacia y eficiencia de los procesos que van a crear valor.
- Comunicación conjunta en los niveles de ambas organizaciones tanto proveedor como empresa, para evitar que no haya problemas de retrasos y costos.
- Seguimiento de los proveedores para entregar productos conformes y suprimiendo verificaciones redundantes.
- Estimular a los proveedores a implementar programas de desarrollo en su desempeño y a participar en otras iniciativas conjuntas de mejora.
- Reconocer y recompensar los esfuerzos y los logros de los proveedores.

4.1.4.12 Recolección, mantenimiento y retención de registros

En esta sección se deberá identificar la documentación SQA que debe conservarse, deberá indicar los métodos e instalaciones que se utilizarán para montar, archivar, proteger, y mantener esta documentación, y se designará el período de retención.

Los registros del aseguramiento de la calidad del software recolectados y archivados deben incluir:

- Informes de tareas.
- Informes de anomalías, discrepancias, defectos no manejados por el mecanismo de informe de problemas.
- Sugerencias a las partes responsables.
- Registro de actividades del aseguramiento de la calidad del software.
- Informes de auditorías.
- Lista de verificación marcadas por los inspectores y auditores.
- Minutas de inspecciones.

Los registros se mantienen durante las etapas de operación y mantenimiento.

4.1.4.13 Formación

En esta sección se deberán identificar las actividades de formación necesarias para satisfacer las necesidades de los PACS.

Actividades como un plan integrado de la calidad:

- Desarrollo de material de formación.
- Implementación de proceso.
- Plan de formación de los miembros de la organización mediante un seguimiento de secuencia durante las distintas fases:

Fases	Descripción
Diagnóstico	Iniciación del proceso, análisis en su totalidad de la situación de la empresa en todas las áreas implicadas. El diagnóstico se realizará en las diferentes actividades, recursos disponibles, documentación, resultados, evolución. Se desarrolla en tres etapas: - Recopilación de la información: Conocer aspectos cualitativos y cuantitativos de la actividad de la empresa. - Análisis de la información: Evalúa el estado actual de la empresa - Conclusiones y discusión de los resultados: Plan de actuación que se realizará durante la implantación
Planificación	Debe coordinar la elaboración del plan de implantación del sistema de calidad e incluyen: plan de acciones, calendario de actividades, revisión de los recursos humanos y económicos. La planificación precisa cuestiones como: - Objetivos que se perciben en la implantación del aseguramiento de la calidad. - Fases que se van a desarrollar. - Responsables para documentar el sistema. - Definición de responsabilidades para el funcionamiento. - Recursos disponibles. - Proceso de auditoría y certificación. Comunicar a la organización lo que se pretende alcanzar con la implantación del aseguramiento de la calidad y el resultado esperado. La información que se debe cubrir sobre las necesidades de quién las recibe y su capacidad para asimilarlas, es decir debe estar adaptada en contenido y extensión. Los datos relevantes sobre las normas a implantar y así como las mejoras en la gestión de calidad.
Documentación del sistema	*Un sistema de aseguramiento de calidad eficaz es aquel que recoge por escrito la forma en que funciona la empresa, por lo tanto el desarrollo del sistema documental es un paso crítico que determinará el éxito de todo el proceso de implantación*. Utilizaremos el manual de calidad, expresa la política de calidad de la empresa siendo un referente en la descripción del sistema de gestión, durante la aplicación y mantenimiento del sistema. Contiene: - Políticas de calidad. - Distribución de responsabilidades. - Relaciones entre los miembros encargados de dirigir, efectuar y revisar las tareas que afectan a la calidad del producto. - Procedimientos e instrucciones del sistema de calidad. - Políticas de revisión, puesta al día y gestión del manual.
Implantación	*La puesta en práctica del sistema se puede realizar básicamente de dos modos:* - *Gradual en que se van asegurando procesos a medida que se van diseñando y documentando los procedimientos del sistema* - *Más ligado a los resultados de los diferentes procesos y actividades en su implantación, que consiste en la puesta en práctica de las actividades de aseguramiento de calidad antes de su documentación definitiva.* Uno de estos modos va a venir determinado por los resultados obtenidos en la fase de diagnóstico del aseguramiento de la

	calidad. En el diagnóstico se establecen actividades y de estas las tareas en la implantación del sistema de calidad, llevadas a cabo mediante la puesta en práctica.
Control y mantenimiento	Una vez alcanzados los objetivos mencionados en las anteriores fases es decir el sistema ya está establecido, es necesario establecer modificaciones. Se determinará qué personas tendrán la responsabilidad de llevar a cabo esta función, para ello se les facilitarán los medios adecuados y materiales para su desarrollo. Se determina dos actividades: 1. Análisis y evaluación del sistema: se realiza en dos momentos distintos de tiempo: ● Previa a la implantación con el objetivo de asegurar una situación de inicio con garantía de éxito desde su partida. ● La implantación con el objetivo de asegurarse mejoras en el sistema de calidad. 2. Realización de auditorías internas con el fin de poner en evidencia las posibles deficiencias para informar a la alta dirección y así poder solventarlas mediante una corrección y prevención.

Tabla 100.Fases del ciclo de vida en la formación del personal

Cabe destacar los siguientes aspectos:

1. Previa planificación y diseño del sistema, se establece una formación en la empresa ante la necesidad e importancia de establecer un sistema de calidad. Es de gran importancia que la dirección conozca: la norma, los procesos a seguir, los recursos a comprometer, tiempo de desarrollo, detección de defectos, plazos de entrega etc.

2. Impartir a todos los miembros de la empresa formación en: resolución de problemas, documentación, trazado y evaluación de procesos, interpretación de normas en las cuales ya están establecidos siguiendo siempre los cometidos específicos.

3. Conjunto del sistema, es necesaria a todo el personal comprometido formación para obtener una correcta implantación en el sistema de calidad.

4. Formación específica a los responsables de auditorías internas, porque de ellos depende el nivel de aprovechamiento que se pueda hacer de la herramienta de gestión.

5. Se deben revisar periódicamente la experiencia y habilidades del personal implicado en la empresa que realizan actividades que afectan a la calidad, estas revisiones determinan las necesidades de formación para un determinado periodo.

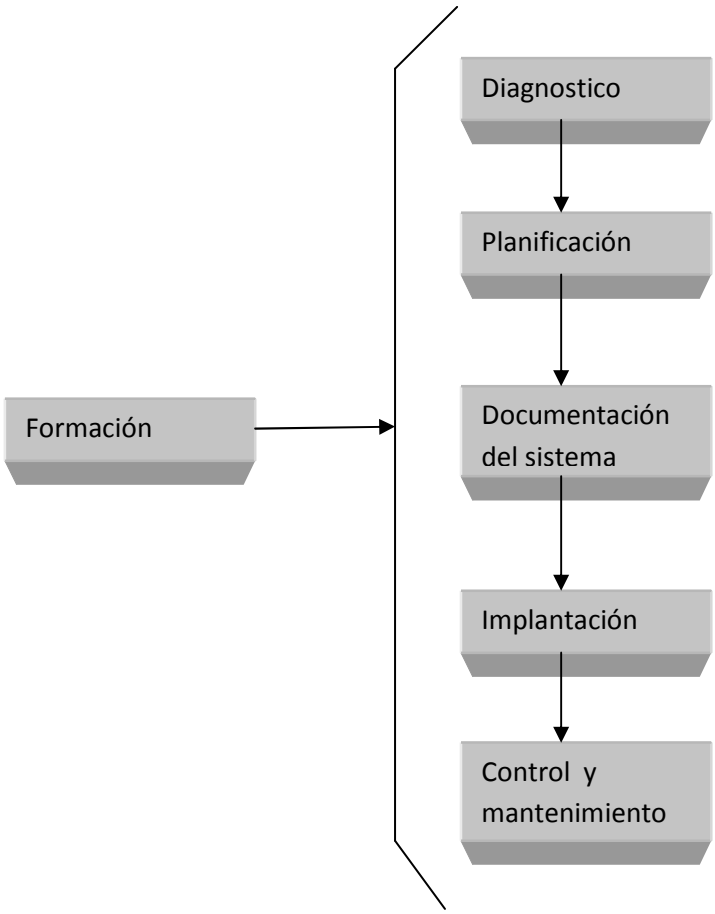


Figura 26.Fases de la elaboración e implantación de un sistema de SQA

4.1.4.14 Gestión de riesgos

En esta sección se expondrán los métodos y procedimientos empleados para identificar, evaluar, supervisar y controlar las zonas de riesgo que surjan durante la parte del ciclo de vida del software cubierto por el PACS.

Un riesgo es algo que puede surgir en el trayecto de un proyecto, según el peor resultado afectaría de manera negativa y significativa. Según la Corporación Lógica más del 70% de todos los proyectos tienen problemas o deterioro severo.

Los factores que ocasionan que un proyecto fracase a la larga aparecen como riesgos cuando se reconoce con prontitud y al reconocer los riesgos tal vez pueda prevenir el fracaso mediante una acción adecuada. Existen dos tipos de riesgo:

- Riesgos que pueden evitarse: Si se identifican con suficiente prontitud, la supresión convierte un proyecto fracasado en exitoso.
- Riesgos que no pueden evitarse: Un proyecto puede detenerse antes de desperdiciar recursos, es decir, para aplicarlos de manera productiva en

otro o se puede modificar el enfoque o agregar personal para minimizar el riesgo.

Cada riesgo identificado debe ser bienvenido en el equipo del proyecto porque se puede empezar a prevenir. Estas actividades que se mencionan en la figura 27 deben llevarse a cabo desde el principio del proyecto.

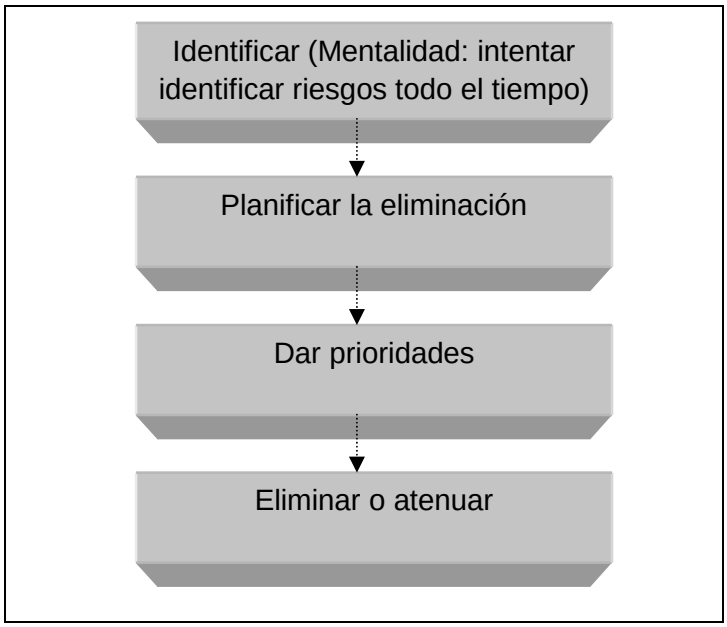


Figura 27.Actividades de administración de riesgo. Adaptado de [Eric J Braude]

La identificación de los riesgos

La identificación de riesgos requiere una mentalidad escéptica similar a la requerida para la inspección; una búsqueda global de defectos en el plan de desarrollo.

- Subestimaciones del tamaño de trabajo.
- Cambios demasiados en los requisitos.
- Falta de habilidad para encontrar una implantación con suficiente eficiencia.
- Deficiencias en las aptitudes del personal.
- Lapso para aprender a utilizar las herramientas: Case
- Deficiencia de los lenguajes.

Tabla 101.Categorías de riesgos

1. Falta de compromiso de la alta administración.
2. Falta de compromiso por parte del usuario.
3. Error al entender los requisitos.
4. Participación inadecuada del usuario
5. Incumplimiento de las expectativas del usuario final.
6. Cambio de alcance y/o de objetivos.
7. Falta de conocimientos o aptitudes requeridas del personal.

Figura 28.En orden de importancia, las fuentes de riesgo. Adaptado de [Eric J. Braude]

En el análisis de la gestión de riesgos se tiene como prioridad colaborar con el equipo del proyecto a tomar una estrategia para evitar los riegos estableciendo lo siguiente:

- Evitar el riesgo.
- Supervisar el riesgo.
- Gestionar el riesgo y los planes de contingencia: suponen que los esfuerzos de reducción han fracasado y que el riesgo se ha convertido en una realidad que debemos afrontar.

Es importante para el gestor de proyectos anticiparse a los riesgos ya que podrían afectar la calidad del software a desarrollar y tomar medidas para evitar esos riesgos.

Todos los resultados obtenidos tras un estudio de los riesgos deben estar debidamente documentados durante el proyecto, junto con la consecuencias cuando ocurre el riesgo, por lo tanto hay que identificar y crear planes para paliar los efectos.

Se puede concebir un riesgo como una probabilidad de que una circunstancia adversa ocurra, los riesgos son una amenaza durante el ciclo de desarrollo. Se establecen los riesgos que se definen a continuación:

- Riesgo del proyecto: Amenazan el plan del proyecto, es decir, identifica problemas en presupuestos, calendarización, personal, recursos, requisitos, impacto sobre el proyecto, complejidad, tamaño, grado de incertidumbre, grado de perdida asociado con cada riesgo estimación etc.

- Riesgo del producto: Afectan a la calidad o al rendimiento del software que se está desarrollando. Si un riesgo se torna real, la implementación se vuelve difícil. Por lo tanto estos riesgos son un problema: diseño, implementación, interfaz, verificación, mantenimiento, incertidumbre, etc.

- Riesgo del negocio: Afecta a la organización que desarrolla o suministra el software, estos riegos ponen en peligro el proyecto o el producto.

  Factores que ponen en peligro los riesgos de negocio:

  1. Riesgo de mercado: Construcción de un producto.
  2. Riesgo de estrategia: Elaboración de un producto que no encaja en la estrategia de la empresa.
  3. Riesgo de ventas: Elaboración de un producto que no saben cómo se venderá.
  4. Riesgo administrativo: Pérdida de apoyo de la alta dirección.
  5. Riesgo presupuestal: Pérdida presupuestaria o del personal asignado.

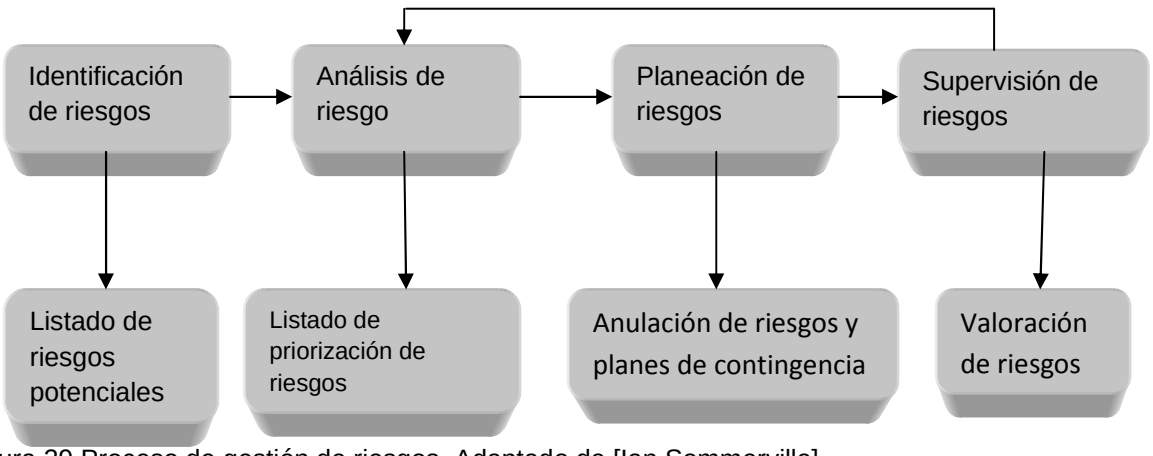


Figura 29.Proceso de gestión de riesgos. Adaptado de [Ian Sommerville]

En la figura 29 consta de las siguientes fuentes de riesgo:

- Identificación de riesgos: Posibles riesgos en la identificación del proyecto, producto y negocio.
- Análisis de riesgo: valorar Probabilidades y consecuencias de los riesgos.
- Planificación de riesgos: Mitigar los riesgos en cuanto la información de los riesgos esté disponible.
- Supervisión de riesgos.

4.1.4.15 Conclusiones generales

- En esta norma lista la verificación de cosas que hay que hacer y buscar, es en particular útil para asegurar que están cubiertas todas las bases de la calidad. Esta norma especifica: quién será el responsable de la calidad, qué documentación se requiere, que técnica se utilizará para asegurar la calidad, qué procedimientos se seguirán para administrar el proyecto: auditorías, revisiones, reuniones etc.

- Hemos utilizado la norma IEEE 829-2005 plan de gestión de configuración del software. La gestión de configuración del software es una disciplina de ingeniería formal, como parte de la gestión general de configuración del sistema, proporciona los métodos y herramientas para identificar y controlar el software a lo largo de su desarrollo y utilización.

- La gestión y configuración del software es el medio por el cual se registran la integridad y la trazabilidad del sistema de software, comunicar y controlar durante el desarrollo y mantenimiento. La gestión y configuración del software es compatible con la reducción del coste total del ciclo de vida del software, proporcionando una base para la medición de productos y proyectos.

- La medición puede ayudar a evaluar la calidad de los productos del ciclo de vida. Evidentemente, el aseguramiento de la calidad no es viable si no existen medios para evaluarla a lo largo del proyecto.

- En el estudio de las iniciativas de aseguramiento de la calidad del software se pone de manifiesto que no tiene sentido trabajar en esta área sin el apoyo de la medición del software. Por ende este apoyo de la medición puede centrarse en las actividades de aseguramiento de la calidad en la mejora de los controles de los productos.

- La medición a nivel de producto como a nivel de proyecto requiere una planificación cuidadosa, su adaptación a los objetivos y a la situación de la empresa y a los principios básicos de la medición.

- Las métricas pretenden contribuir a la evaluación de diversos aspectos de la calidad del software a lo largo de su ciclo de vida, así como permitir la prevención de problemas sugiriendo reglas de diseño o programación a los desarrolladores.

- El principal problema de las métricas para evaluar la calidad o alguno de sus aspectos es que no queda clara y valida su relación con lo que se pretende evaluar, ya que en líneas generales carecen de experimentación real y de una validación que avale su utilidad para el control de productos o de procesos.

- Se pone de manifiesto la utilización de herramientas entre ellas las herramientas CASE para ayudar al diseño y evaluación de mejoras de dicho proceso. En este ámbito se propone la utilización de la técnica dentro de un marco de referencia como puede ser un modelo de madurez de proceso (CMM).

4.2 Especificación de los requisitos software IEEE 830 – 1998

4.2.1 Introducción

En este apartado se hace una descripción de lo que va a contener el documento.

Para ello definiremos la importancia de los requisitos:

"Para que un esfuerzo de desarrollo de software tenga éxito, es esencial comprender perfectamente los requisitos del software. Independientemente de lo bien diseñado o codificado que esté un programa, si se ha analizado y especificado pobremente, decepcionará al usuario y desprestigiará al que lo ha desarrollado." [Roger S. Pressman Ingeniería del Software Mc Graw Hill 1995]

"Una condición o necesidad del usuario para resolver un problema o alcanzar un objetivo".(Std 610.12-1900,IEEE62).

"Condición o capacidad que debe estar presente en un sistema o componente del sistema para satisfacer un contrato, estándar, especificación u otro documento formal" (Std 610.12-1900,IEEE62).

Entender el problema de los usuarios en su cultura, lenguaje y construir el sistema que resuelve sus necesidades.

Capacidad o condición que debe realizar un sistema dependiendo de la necesidad del usuario o bien estipulada en un contrato, estándar especificación u otro documento formalmente impuesto al inicio del proceso.

Cabe destacar:

El contrato: Es un documento legal obligatorio en el cual están implicados el cliente y proveedor. Se incluyen los requisitos técnicos, requisitos de la organización, costo y tiempo para un producto. Un contrato también puede contener la información informal pero útil como los compromisos o expectativas de las partes involucradas.

Cliente y Proveedor: Establecer las bases para un acuerdo entre los clientes y los proveedores en lo que se refiere al producto del software en que se va realizar, descripción completa de las funcionalidades para una buena especificación de requisitos del software, esto es muy importante ya que ayudará a los usuarios potenciales a satisfacer sus necesidades, y esto conlleva modificaciones que pueden reducir el esfuerzo del desarrollo y revisiones de los requisitos ya que puede revelar omisiones, malentendidos y contradicciones a principios del ciclo de desarrollo cuando los problemas son más fáciles de corregir.

4.2.1.1 Naturaleza del SRS

El SRS son especificaciones para un producto del software en particular, programa, o juego de programas que realizan ciertas funciones en un ambiente específico. El SRS puede escribirse por uno o más representantes del proveedor, uno o más representantes del cliente, o por ambos [IEEE 830 – 1998 especificación de los requisitos software]

Naturaleza del SRS: SRS son especificaciones para un producto de software
Los problemas básicos que se presentan al escribir un SRS van dirigidos a lo siguiente:

Los requisitos de baja calidad

Antes del proceso de SRS el cliente tiene una idea para un sistema, para una mejora de los procesos, o para un problema a resolver. En este punto, cualquier concepto inicial de un sistema puede ser impreciso y no estructurado.

Los requisitos a menudo se entremezclan con las ideas y sugerencias de diseños posibles. Estos requisitos de baja calidad se expresan a menudo en la iniciación de documentos similares a lo siguiente:

a) Concepto de las operaciones.

   Este tipo de documento se centra en las metas, objetivos, y en general se desea que las capacidades del sistema sean posibles sin indicar cómo el sistema se implementará para alcanzar realmente los objetivos.

b) Concepto de sistema.

   En este tipo de documento no sólo se incluye el concepto de información de operaciones, sino que también se incluye una interfaz de diseño preliminar para el sistema y otros requisitos explícitos.

c) Comercialización de las especificaciones.

   Este tipo de documento incluye una lista de características (a menudo en formato de viñeta) para un nuevo sistema o sistemas y determinar el alcance de las funciones y sus prioridades (que sean obligatorios o muy convenientes) para proporcionar una ventaja en el mercado. También incluye un marco o límite que define cómo el nuevo sistema debe interactuar con los sistemas existentes. Un análisis de costo / beneficio y plazos de entrega exigida.

d) Solicitud de propuesta (RFP).

   En algunos casos, una solicitud de propuestas estará preparada. Esto puede incluir uno o más de los documentos antes de iniciar. Su finalidad será la de solicitar ofertas para la consideración de diversas fuentes para

construir un sistema, o simplemente pueden necesitar asistencia para generar documentos del sistema de iniciación.

e) Interfaces externas del sistema.

La definición de todas las interfaces externas al sistema, literalmente, o por de referencia, es uno de los más importantes (pero más a menudo es pasado por alto) las actividades a llevarse a cabo antes de la generación de la SRS. Todos los elementos conocidos de cada interfaz definida por separado deben ser descritos. Esta información puede ser incluida en el SRS si no es demasiado voluminosa.

Sin embargo, en la mayoría de los casos es mejor tener un sistema externo de control de documentos de interfaz (ICD). Hay muchos tipos de interfaces posibles externos al sistema y un solo sistema puede tener diversas interfaces de diferentes tipos.

Una característica del producto que los clientes exigen es la fiabilidad.

Ireson (1996) afirma, al introducir el concepto de entorno operativo, que la fiabilidad es la habilidad o capacidad de un producto para ejecutar la función para la que está especificada en un entorno, previamente designado, y durante un tiempo mínimo de veces u ocasiones, con lo cual se hace referencia a condiciones y entornos operativos específicos. Juran y Gryna (1993) analizan los principales elementos del programa:

- Establecer objetivos de fiabilidad. - Determinar modelos de fiabilidad. - Asignación y reparto de objetivos de fiabilidad. - Análisis de estrés. - Predicción de fiabilidad. - Análisis de efecto y modelo de fallo. - Identificación de componentes críticos. - Revisión de diseño. - Selección de proveedores. - Control de fiabilidad en la producción. - Ensayos de fiabilidad. - Informes de fallos y acciones correctoras.

Tabla 102.Elementos principales de un programa

4.2.1.2 Ambiente del SRS

Desde que el SRS tiene un papel específico en el proceso de desarrollo de software, el que define el SRS debe tener el cuidado para no ir más allá de los límites de ese papel.

Requisito	Descripción
Normalizado	Los requisitos no deben solaparse (es decir, no se refieren a otros requisitos o las capacidades de los otros requisitos).
Conjunto vinculados	Relaciones explícitas, deben ser definidas entre las necesidades individuales para mostrar cómo los requisitos están relacionados con la forma de un sistema completo.
Completos	Una SRS debe incluir todos los requisitos definidos por el cliente, así como los necesarios para definir el sistema.
En consonancia	Una SRS contenida debe ser coherente y no contradictoria en el nivel de detalle, el estilo de las declaraciones de los requisitos, y en la presentación del material.
Limitado	Los límites, el alcance y el contexto para el conjunto de requisitos deben ser identificados.
Modificable	La SRS debe ser modificable. Claridad y requisitos que no se superponen para contribuir en ello.
Configurables	Las versiones se deben mantener en el tiempo y en todas las instancias de la SRS.
Granular	Nivel de abstracción para el sistema para el que se ha definido.

Tabla 103.La colección de los requisitos debe tener las siguientes propiedades

Una SRS correctamente escrita proporciona beneficios de todas las fases posteriores del ciclo de vida de varias maneras diferentes. La SRS documenta el conjunto completo de capacidades del sistema.

- Garantía para el cliente, el equipo técnico comprende las necesidades del cliente y responde a ellos.
- Una primera oportunidad para la retroalimentación bidireccional entre el cliente y el equipo técnico.
- Métodos para el cliente y el equipo técnico para identificar los problemas y malos entendidos, si bien es relativamente barato poder corregir.
- A base de un sistema de clasificación para establecer que el sistema cumple con las necesidades de los clientes.
- Protección del equipo técnico, proporcionando una base para las capacidades del sistema y una base en la determinación de la construcción del sistema completo.
- El apoyo a la planificación de programas, diseño y desarrollo.
- Ayuda en la evaluación de los efectos de los cambios de los requisitos inevitables.

Tabla 104.Ventajas que proporciona una SRS correctamente escrita

4.2.1.3 Características de un buen SRS

- **Una E.R debe ser correcto**:

  Un SRS es correcto si, y sólo si, cada requisito declarado se encuentra en el software.

  Una definición de corrección indica que “un conjunto de requisitos software es correcto sólo si todos los requisitos contenidos representan

algo que es requerido para la construcción del sistema y no hay errores que afecten al diseño".

Un aspecto importante a tener en cuenta acerca de la corrección es que no se puede establecer a priori, sino que depende fundamentalmente del usuario final del sistema representado. Quien debe decidir si una especificación es correcta o no es el cliente que solicita el sistema. Por eso la corrección de una especificación debe ser verificada a través de:

- La revisión y aceptación del usuario.
- Problemas habituales en este ámbito, son la omisión de información relevante, por parte de los usuarios y la adición de requisitos no solicitados, por parte de los equipos de análisis y desarrollo.

- **Una E.R debe ser Inequívoca:**

La especificación de requisitos software debe contener una serie de requisitos bien definidos que proporcionan un concepto definido del futuro sistema.

- **Una SRS debe ser completa**

D*eben considerarse todos los aspectos relacionados con la funcionalidad, rendimientos, restricciones, etc. Todo esto debe estar debidamente documentado. No se debe dejar nada para analizar más adelante en el proyecto.*

Se hace un esfuerzo para que cada requisito específico sea autocontenido pero pocas veces es posible en la práctica. Ya que los requisitos con frecuencia hacen referencia a otros requisitos. Que un conjunto de requisitos esté completo asegura que no hay omisiones que comprometan los requisitos establecidos.

- Incluye todos los requisitos significativos del software, ya sean relativos a la funcionalidad, ejecución, atributos de calidad, interfaces externas.
- Define la respuesta del software a todas las posibles clases de datos de entrada y en todas las posibles situaciones. Es importante el especificar las respuestas tanto para entradas válidas como no válidas.
- Está conforme cualquier norma de especificación que se deba cumplir. La adaptación a una norma puede implicar que si una sección particular del estándar no es aplicable al desarrollo del que se trata, la SRS debe incluir el correspondiente número de sección del estándar y una explicación que justifique su no aplicación o, al menos se pone la expresión no aplicable para tener constancia de que el apartado no está vacío por error.

Figura 30.Descripción de la SRS completa

- **Una SRS debe ser consistente**

  No deben existir conflictos en los términos y expresiones utilizados.

  Es fácil de leer y entender, su redacción debe ser simple y clara para aquellos que vayan a utilizarla en un futuro remoto.

  Una ERS es consistente si y sólo si ningún conjunto de los requisitos descritos en ella son contradictorios o entran en conflicto.

- Dos o más requisitos pueden describir al mismo objeto real pero utilizan distintos términos para asignarlos. - Las características especiales de objetos reales pueden estar en conflicto. - Conflicto lógico o temporal entre dos acciones determinadas.

Figura 31.Tipos de conflictos.

Cuando el número de requisitos específicos crece, la consistencia se puede volver algo difícil de lograr.
La organización orientada a objetos en los requisitos ayuda a evitar consistencias mediante la agrupación de los requisitos específicos en las clases y descomposición en formas muy sencillas. Sin embargo, esto no es una garantía de consistencia y por ello las inspecciones en los requisitos se verifican con otras características mencionadas.

- **Una SRS debe delinear que tiene importancia y/o estabilidad**

  Una SRS debe delinear la importancia y/o estabilidad si cada requisito en ella tiene un identificador para indicar la importancia o estabilidad de ese requisito en particular.

  Los requisitos deben tener establecido un orden de prioridad basado en su importancia para la aplicación o, alternativamente, una clasificación en función de su estabilidad, su resistencia a la volatilidad. Esta ordenación debe permitir que los desarrolladores puedan decidir mejor los requisitos que puedan quedar.

Grados	Descripción
Grado de estabilidad	Se refiere al número de modificaciones que se esperan en los requisitos los cuales se basan en conocimientos de eventos futuros que afectan a la organización, métodos y usuarios que avalan el sistema del software.
Grado de necesidad	Podemos distinguir los requisitos en tres categorías: - Esencial: Factor útil que debe tenerse en mente si el tiempo para el proyecto comienza a ser apremiante. - Condicional: Son requisitos que fortalecerían el producto del software. - Opcional: Conlleva funciones presentes en una clase que tal vez puede ser redundante o no puede valer la pena.

Tabla 105.Grados de Estabilidad

- **Una SRS debe ser comprobable**

  Debe ser posible validar un requisito cuando se prueba que se ha implementado de manera apropiada. Los requisitos que se pueden probar se llaman comparables. Los requisitos no comparables tienen poco valor práctico.

  Una SRS es fácil de verificar si cualquier requisito al que se haga referencia se puede verificar fácilmente, esto quiere decir, si existe un procedimiento finito y eficaz en coste para que una persona o máquina compruebe que el software satisface dicho requisito.

- **Una SRS debe ser modificable**

  Una SRS es modificable si su estructura y estilo permiten que cualquier cambio necesario en los requisitos se pueda desarrollar con facilidad, completitud y consistencia.

  - Tener una organización coherente y manejable, tabla de contenido, índice y referencias cruzadas. De esta manera es sencillo saber dónde hay que modificar el documento cuando hay un cambio en los requisitos
  - No ser redundante, es decir, que el mismo requisito no aparezca en más de una SRS. Un cambio supone modificar el documento en un solo sitio.

  Figura 32.Implicación en la SRS modificable

  La redundancia en sí no es un error, pero puede conducir a problemas. La redundancia puede ayudar a la legibilidad de la SRS, pero provocará problemas cuando haya que actualizar el documento.

  La redundancia es difícil de evitar, para solventarlo lo mejor es crear referencias cruzadas entre los requisitos y los términos empleados para definirlos facilitando así las modificaciones en la SRS.

- **Una SRS debe ser Identificable**

  Se dice que una SRS facilita las referencias con otros productos del ciclo de vida si establece un origen claro para cada uno de los requisitos y si posibilita la referencia de estos requisitos en desarrollos futuros o en incrementos de la documentación.

  - Cada requisito debe identificarse con algún número, letra o secuencia alfanumérica.
  - Tipos de rastreabilidad:

    1. Referencias hacia atrás: Depende de que los requisitos referencien explícitamente sus fuentes en documentos previos.

2. Referencias hacia delante: Depende de cada requisito de la SRS tenga un nombre o un número de referencia único que sirva para identificarlo en futuros documentos.

- Cuando un requisito de la SRS representa un desglose o una derivación de otro requisito, se deben facilitar tanto las referencias hacia atrás como hacia adelante en el ciclo de vida. - Las referencias hacia delante de la SRS son especialmente importantes para el mantenimiento del software. - Cuando el código y los documentos son modificados, es importante comprobar el conjunto total de requisitos que pueden verse afectados por las modificaciones.

Tabla 106.Aspectos importantes a tener en cuenta para identificar el origen y consecuencias de cada requisito.

- **Una SRS debe ser trazable**

Traza de requisitos funcionales

Cuando los requisitos cambian, lo cual se puede suponer, esto es aún más difícil. La capacidad de hacer corresponder cada requisito con sus partes relevantes de diseño e implementación se denomina trazabilidad. Una manera de ayudar a lograr esto es mantener una correspondencia de cada requisito específico funcional con una función específica del lenguaje a usar.

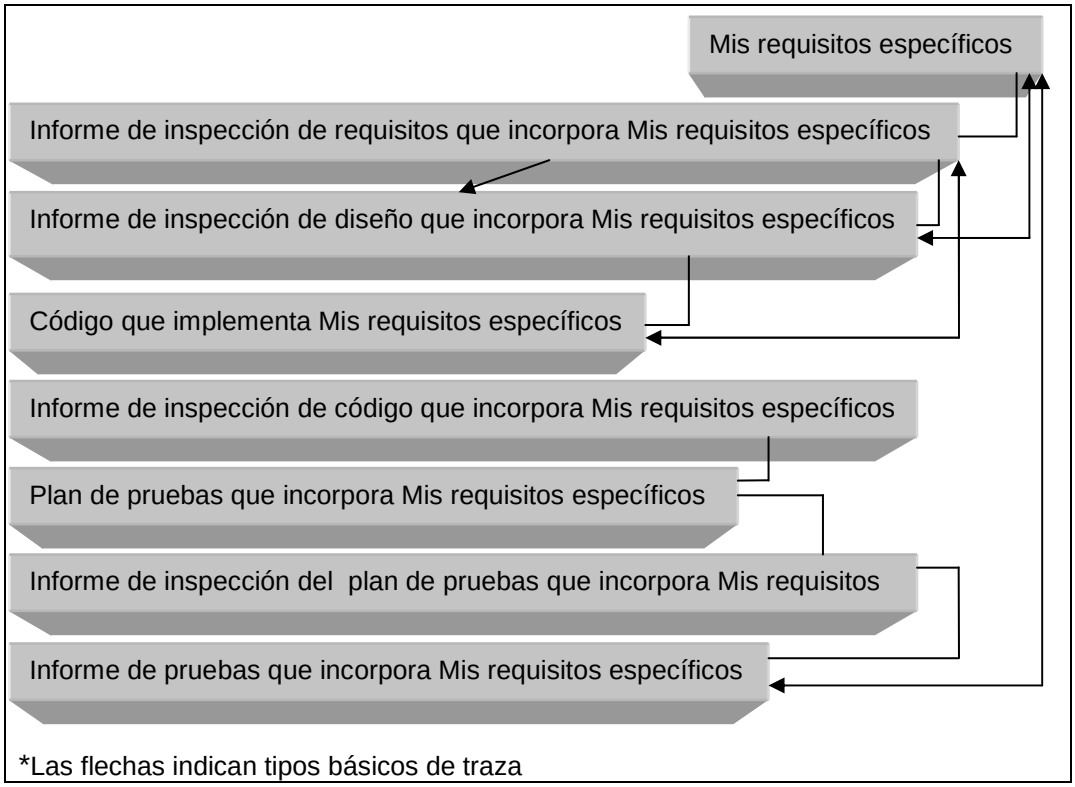


Figura 33.Traza de requisitos específicos. Adaptado de [Eric J. Braude]

La figura 33, muestra las partes del proyecto que deben relacionarse con una trazabilidad verdadera. Lograr y mantener este grado de trazabilidad durante el desarrollo es un reto importante.

Conforme avanza el proyecto, el documento de requisitos debe permanecer consistente con el diseño y el código, los desarrolladores tienden a evitar actualizar el documento de requisitos al hacer cambios al mismo código fuente debido al gran esfuerzo que implica.

Traza de requisitos no funcionales

En la etapa de requisitos es especificar lo más claro posible los requisitos no funcionales. Un objetivo de la etapa de diseño es aislar cada requisito no funcional en un elemento de diseño separado.

En el caso de los requisitos de rendimiento se intenta aislar las unidades de procesamiento más lentas. Cada función que afecta los requisitos de rendimiento se acompaña por los comentarios no funcionales que se pueden inspeccionar.

De manera similar, al especificar las restricciones de almacenamiento se identifican las funciones que generan el mayor almacenamiento.

La investigación muestra que un porcentaje relativamente pequeño de funciones en una aplicación son responsables de la mayor parte del procesamiento y, por lo tanto, la búsqueda de unos cuantos consumidores de tiempo puede ser fructífera.

En las fases de diseño e implementación se buscan los componentes típicos que consumen tiempos en los cálculos de enlace. Para validar los requisitos no funcionales, se hace corresponder cada uno de ellos con un plan de pruebas.

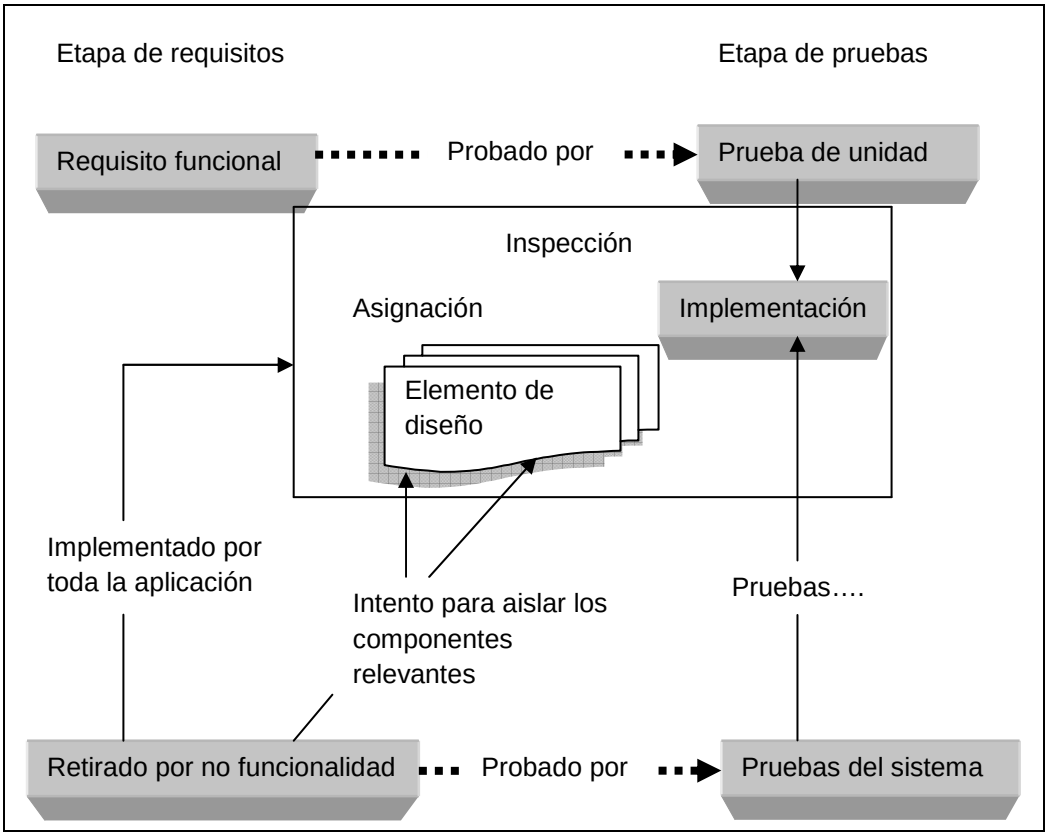


Figura 34.Traza y prueba de requisitos no funcionales y funcionales. Adaptado de [Eric J. Braude]

La figura 34 muestra la relación típica de los requisitos funcionales y no funcionales con la implementación y las pruebas, así como el hecho de que varios elementos pueden influir en los requisitos no funcionales, y que se requieren pruebas del sistema o de integración para validar los requisitos no funcionales, es decir, verificarlos antes de la ejecución puede ser difícil.

4.2.1.4 La preparación de los Joins o conjunta del SRS

El desarrollo de un software solamente debería iniciarse cuando el cliente y el proveedor estén de acuerdo acerca de lo que el software deberá hacer. En otras décadas era corriente que los clientes sostuvieran que el proveedor deberá resolver sus problemas de calidad.

Hoy en día existe una creciente concienciación de que estos problemas son un compartimento compartido basado en:

- Establecimiento de una confianza mutua.
- Definición de la calidad tanto en términos de necesidades de los clientes como de especificaciones.
- Intercambio de datos esenciales.
- Comunicación directa tanto a nivel técnico como a nivel comercial.

Los primeros pasos en el despliegue de las métricas del servicio al cliente son la necesidad de identificar los requisitos para las métricas y definir métricas específicas que satisfagan estos requisitos.

- Eficacia del proceso de servicio al cliente. ¿La salida del proceso acude al encuentro de las necesidades del cliente? Una aproximación a esta métrica es cuantificar los defectos que produce el proceso. - Eficiencia del proceso de servicio al cliente. ¿El proceso utiliza convenientemente los recursos? Una aproximación a esta métrica es cuantificar los defectos internos del proceso. - Eficacia del proveedor. ¿Las entradas al proceso satisfacen sus necesidades? Una aproximación a esta métrica es cuantificar los defectos entrantes.

Tabla 107.Las métricas aluden a los procesos mediante estos términos

Las métricas mencionadas en la tabla 107, se centran en los defectos. Un defecto es algo que no satisface o excede los requisitos del cliente, el negocio o el proceso. Aunque el foco sobre los defectos es, de una forma primaria, en torno a los defectos internos, más que en los defectos que afectan al cliente, el énfasis en los defectos obedece a una aproximación preventiva, eliminando problemas antes de que tengan algún efecto en el servicio prestado al cliente.

Cuando se definen las métricas, es importante asegurar que los datos con los que se surtirán se pueden conseguir de forma fácil y legible. El equipo encargado de ello debería pensar en cuáles son las fuentes, así como los métodos de recolección, como se identificarán los defectos, la cantidad de esfuerzo requerido y los tiempos de ciclo para la recogida de los datos.

Todas estas características clave deberían ser representadas con métricas, incluso aunque todas ellas no puedan ser implantadas en una primera fase.

El equipo encargado de las métricas debería pensar en la forma en que cada uno de los métodos puede afectar a las conductas y modificarlas, si es preciso, para asegurar que las métricas sean capaces de dirigir hacia la conducta deseada.

Las mediciones de calidad de servicio al cliente apelan a cuatro áreas que afectan directamente a los clientes:

Requisito	Descripción
Requisitos de los clientes	Se presentan los siguientes aspectos: - Asociación con los clientes. - Aseguramiento de la calidad. - Fiabilidad. - Empatía. - Durabilidad. - Capacidad de respuesta.
Requisitos de tareas	Se presentan los siguientes aspectos: - Opciones de suministrador. - Opciones de comercialización. - Opciones operativas. - Opciones departamentales. - Opciones interfuncionales.

	- Opciones horizontales. - Opciones de producción. - Opciones de entrega. - Opciones de consumo.
Requisitos organizativos	Se presentan los siguientes aspectos: - Compromiso en la gestión. - Preparación y formación. - Definición de roles y responsabilidades. - Comunicaciones. - Proceso de establecimiento de objetivos.
Requisitos de control de procesos	Se presentan los siguientes aspectos: - Herramientas de medición. - Herramientas de evaluación. - Herramientas de mejora. - Información. - Asignación de recursos. - Planificación. - Mecanismos de retroalimentación. - Herramientas de monitorización del entorno.

Figura 35.Áreas que afectan directamente a los clientes

Cada una estas cuatro áreas se desagrega a su vez en cinco niveles.

Nivel de calidad en el servicio	Áreas potenciales de impacto
Conveniente para el uso	Cliente.
Conveniente para el estándar	Procesos operativos.
Conveniente para la demanda	Tiempo de ciclo.
Conveniente para la planificación	Planificación de las variables del servicio al cliente
Conveniente para el control	Protección de los requisitos del cliente.

Tabla 108.Impactos de la calidad en el servicio al cliente. Adaptada fuente [Joseph M. Juran]

Incluyen métricas objetivas, como los índices de errores y niveles de defecto, y medidas subjetivas, como los posicionamientos de calidad en términos competitivos. En ambos casos, la utilidad de los datos se determina por la eficacia de las definiciones de lo que constituye un defecto o un error, cómo se diseñan y se implantan los cuestionarios y las encuestas a clientes, y cómo se analizan los datos y se utiliza para controlar, mejorar y priorizar.

4.2.1.5 La evolución del SRS

La ERS necesitará ser modificadas a medida que progresa el producto software.

Es casi imposible especificar algunos detalles en el momento en que se inicia el proyecto; se realizarán cambios adicionales como consecuencia de haber encontrado deficiencias, defectos que se descubren a medida que el producto evoluciona.

- El requisito deber ser especificado de la forma más completa posible, aún en el caso en el que se prevean de manera inevitable revisiones en el proceso de desarrollo, es decir, especificar lo mejor posible en la SRS, de manera que sirva de base para el diseño posterior.
- Debe comenzar un proceso formal de cambio para identificar, controlar, seguir e informar de cambios proyectados, tan pronto como sean identificados. Los cambios aprobados en los requisitos deben ser incluidos en la SRS de la manera que permita lo siguiente:
    - Suministrar una revisión adecuada en la traza de las modificaciones.
    - Permitir una revisión de fragmentos actuales y reemplazados de la SRS.

Tabla 109.Consideraciones a tener en cuenta en un proceso de evolución del software.

La flexibilidad de los sistemas es una de las principales razones por las que más software se incorpora a los sistemas grandes y complejos. Una vez que se decide adquirir hardware, es muy difícil hacer cambios en su diseño, se pueden hacer cambios al software en cualquier momento durante o después del desarrollo del sistema.

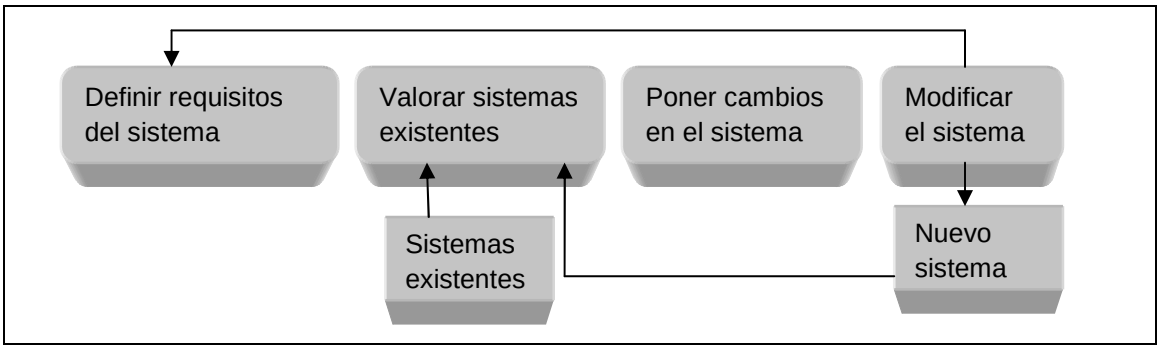


Figura 36.Evolución del sistema. Adaptado de [Ian Sommerville]

En la figura 36, en la cual el software cambia continuamente durante su periodo de vida como respuesta a los requisitos cambiantes y necesidades del usuario.

4.2.1.6 Prototipos

Los prototipos juegan un papel muy esencial en la creación de requisitos.

- *Un prototipo es un pequeño programa parecido al software solicitado que sirve de ejemplo, muestra o modelo para que el cliente vaya especificando sus requisitos en forma progresiva junto con el desarrollador.*

- *El prototipo es útil para:*

    1. *Que el cliente/usuario vea y describa más fácilmente las funcionalidades que desea.*
    2. *Prever aspectos de la conducta del sistema, haciendo que el SRS sea más completo y preciso.*

3. *Reducir la cantidad de cambios durante las etapas de diseño o desarrollo.*

El objetivo del prototipo es derivar y validar los requisitos esenciales, al mismo tiempo las opciones de implementación.

- El área de la aplicación no está bien definida, bien por su dificultad, bien por la falta de tradición en su informatización. - El coste de rechazo de la aplicación por los usuarios, por no cumplir sus expectativas, es muy alto. - Es necesario evaluar previamente el impacto del sistema en los usuarios y la organización.

Tabla 110.Utilidad de los prototipos

Los prototipos permiten al desarrollador crear un modelo del software que debe ser construido. Existen distintos tipos de prototipos en orden decreciente mediante su frecuencia de uso.

Tipo	Descripción
Prototipado de la interfaz de usuario	Para asegurarse que se encuentra bien diseñada y satisfacer la necesidades de quienes deben utilizarlo. Se distinguen dos niveles: - Nivel más sencillo: Incluye modelos de pantallas en papel (publicidad, programas sencillos de dibujo, presentación (Ejemplo, PowerPoint). - En el nivel más sofisticado: Puede tratarse de simulaciones de la interfaz muy elaborada que son cada vez más fáciles de construir con los actuales entornos de programación y facilidades de los componentes en el diseño de interfaz de usuario.
Modelos de rendimiento	Para evaluar el rendimiento de un diseño técnico, especialmente en aplicaciones críticas. Estos modelos tienen un carácter técnico y por lo tanto no son aplicables al trabajo de análisis de requisitos.
Prototipo funcional	Cada vez más demandado. En vez de seguir el procedimiento habitual, es decir, tirar el prototipo una vez probado y empezar a desarrollar la aplicación; el prototipo supone una primera versión del sistema con funcionalidad limitada. A medida que se comprueba si las funciones implementadas son las apropiadas, se corrigen, refinan o se añaden otras nuevas, hasta llegar al sistema final.

Tabla 111.Tipos principales de prototipos

La cualidad esencial de un prototipo es que puede ser elaborado más rápidamente que la aplicación correspondiente. Por esta razón se suele hablar de proceso de “diseño rápido”.

Diseño rápido: Se centra en una representación de aquellos aspectos del software que serán visibles al usuario (por ejemplo, entradas y formatos de las salidas). El diseño rápido lleva a la construcción de un prototipo. El prototipo es

evaluado por el cliente y el usuario y utilizado para refinar los requisitos del software a ser desarrollado.

- En el nivel inferior se encuentran herramientas comunes y disponibles como: hoja de cálculo, informes, programas de dibujo o de presentación. El usuario puede observar cómo quedará la entrada/salida de la aplicación cuando esté finalizada. - En un nivel más evolucionado se sitúan algunos gestores de bases de datos y sistemas que permiten prototipos más sofisticados que incluyen no sólo interfaces sino también prototipado del manejo de datos. - Posibilidades de prototipado que proporcionan las herramientas CASE o determinados generadores de aplicaciones. Se pueden reutilizar las plantillas de pantalla, etc. Encontramos componentes, utilidades y bibliotecas de desarrollo de interfaces de los lenguajes y entornos actuales para construir versiones iniciales o prototipos evolutivos de la aplicación.

Tabla 112.Herramientas empleadas en el prototipado

Al igual que todos los enfoques al proceso de desarrollo del software, el prototipado comienza con la captura de requisitos. Los desarrolladores y clientes se reúnen y definen los objetivos globales del software, identifican todos los requisitos que son conocidos, y señalan áreas en las que será necesaria la profundización en las definiciones. Luego de esto, tiene lugar un "diseño rápido".

4.2.1.7 Generando el diseño de SRS

El generar el diseño de SRS implica:

- *Aunque el SRS no constituye un documento de diseño, implícitamente está diciéndole a los desarrolladores lo que se espera que ellos diseñen el establecer restricciones.*
- *Algunas situaciones especiales pueden inducir requisitos estrictos de diseño.*
- *Estándares de aseguramiento de calidad del software.*

La etapa de la implementación del desarrollo de software es el proceso de convertir una especificación del sistema en un sistema ejecutable. Implica los procesos de diseño y programación del software, pero si se usa un enfoque evolutivo de desarrollo también puede implicar un refinamiento de la especificación del software.

El proceso de diseño puede implicar el desarrollo de varios modelos del sistema con niveles de abstracción. Mientras se descompone un diseño, se descubren errores y descuidos en las etapas previas. Esta retroalimentación permite mejorar los modelos de diseño previo.

Una etapa para la siguiente etapa es la salida de cada actividad de diseño. Esta especificación puede ser abstracta y formal, realizada para aclarar los requisitos, o puede ser una especificación para determinar qué parte del

sistema se va a construir. El resultado final del proceso son especificaciones precisas de los algoritmos y estructura de datos a implementarse.

Las actividades específicas del proceso de diseño son:

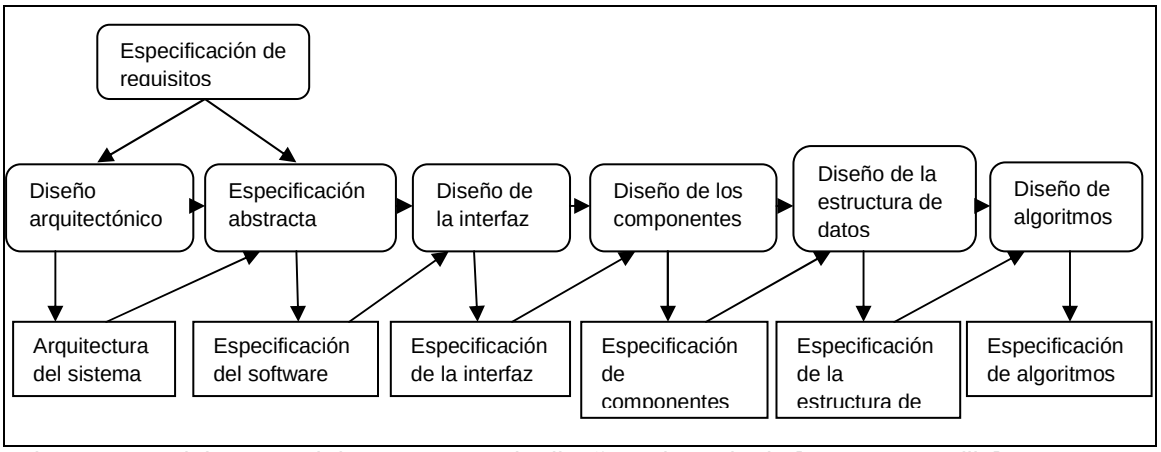


Figura 37.Modelo general de un proceso de diseño. Adaptado de [Ian Sommerville]

En la figura 37, figura un modelo de este proceso que muestra las descripciones de diseño que pueden producirse en varias etapas de diseño.

- Diseño arquitectónico: Los subsistemas que forman el sistema y sus relacionan se identifican y documentan.
- Especificación abstracta: Para cada sistema se produce una especificación abstracta de sus servicios y las restricciones bajo las cuales debe funcionar.
- Diseño de la interfaz: Esta especificación de la interfaz debe ser inequívoca ya que permite que el subsistema se utilice sin conocimiento de su funcionamiento.
- Diseño de componentes: Se asignan servicios a los componentes y se diseñan sus interfaces.
- Diseño de la estructura de datos: Se diseña en detalle y especifica la estructura de datos utilizada en la implementación del sistema.
- Diseño de algoritmos: Se diseñan en detalle y especifican los algoritmos utilizados para proporcionar servicios.

Tabla 113.Explicación de los componentes principales de la figura 37

4.2.1.8 Generando los requisitos del proyecto en el SRS.

El generar los requisitos del proyecto en el SRS implica:

- *Implícitamente establece restricciones sobre la planeación y administración del proyecto correspondiente.*
- *El SRS da origen a otros documentos (por separado) relacionados con el ciclo de vida de un software.*

    - *Estimación de costos (¿cómo calcularlos?).*
    - *Fechas de entrega.*
    - *Reportes de avances.*
    - *Métodos de desarrollo.*

- *Aseguramiento de la calidad.*
- *Criterios de validación y verificación.*
- *Procedimientos de aceptación.*

Los requisitos específicos se colocan bajo el control de la configuración.

	Estado después del borrador inicial	Resultado de actualización después de los requisitos del cliente	Resultado de actualización después de los requisitos específicos
Eventos importantes	inicial	Más detallados	Más detallados
Riesgos	identificar	Eliminar riesgos identificados	Eliminar riesgos identificados
Programación	Nivel muy alto	Tiempos preliminares del proyecto	Más detalle: Muestra las clases y el método de desarrollo
Personal	Designar ingenieros para los requisitos específicos	Designar ingenieros para análisis de requisitos específicos	Designar arquitectos de software
Estimación de coste	Estimaciones ordinarias	Primeras estimaciones basadas en el contenido del trabajo	Estimaciones mejoradas basadas en punto de función más especificadas o en la experiencia con requisitos individuales

Tabla 114.Actualización del proyecto al completar los requisitos específicos. Adaptado de [Eric J. Braude]

Cuando la lista de requisitos crece mucho, con facilidad surgen inconsistencias.

Clasificar los requisitos por clases, las clases por paquetes y los paquetes por subpaquetes, y así sucesivamente se convierte en una necesidad. Los paquetes por lo común, corresponden a subsistemas en toda la organización de la aplicación.

La administración de proyectos consiste en gestionar la producción de un producto dentro del tiempo dado y los límites. Como esto requiere recursos humanos, la administración del proyecto involucra no sólo la organización técnica y las habilidades organizativas, sino el arte de administrar personas.

El administrador de proyectos puede controlar de alguna manera los siguientes factores:

- El costo total del proyecto.
- Las capacidades del producto.
- La calidad del producto.
- La duración del proyecto.

Figura 38.Variables de la administración del proyecto.

Los componentes de la administración de proyectos comprenden:

- Estructura: Elementos organizacionales involucrados.
- Proceso administrativo: Responsabilidades y supervisión de los participantes.
- Proceso de desarrollo: Métodos, herramientas, lenguajes, documentación y apoyo.
- Programa: Tiempos en los que deben realizarse las porciones de trabajo.

- Costo: Pueden estar fijos de antemano, muchas veces hay cierta flexibilidad.
- Capacidad: Tampoco son entidades fijas que aparecen. Por ejemplo, el cliente puede estar de acuerdo en eliminar un requisito si hacerlo disminuye el 15% la duración del proyecto, es decir, trueque entre capacidad y programa.
- Calidad: Las metas de calidad pueden variar.
    - Cuando las metas de calidad se estiman demasiado bajas: Se crea un desequilibrio en los costos a corto y largo plazos debido al doble trabajo e insatisfacción del cliente.
    - Cuando las metas de calidad es estiman demasiado altas: El costo de encontrar hasta el detalle más pequeño puede ser prohibitivo.
- Programa: En ocasiones, se pueden negociar las fechas de terminación. Por ejemplo, un cliente está dispuesto a cambiar una fecha de entrega si el producto resultante tiene tantas capacidades que es probable que se capte en el mercado.

Figura 39.Variables principales: costo, capacidad, calidad y programa

El coste de un proyecto es de interés continuo y vital. Con el precio de producción equivocado, incluso el producto más fantástico puede ser un desastre. La estimación de costo más sencilla es la que proporciona un costo fijo desde el principio, sin permitir desviaciones en ninguna circunstancia.

Aunque las organizaciones muy competentes tienen suficientes aptitudes para cambiar las variables restantes (capacidad, programa de tiempos y calidad) para cumplir con un costo predeterminado, la rigidez absoluta en el costo no siempre se adopta.

El proceso de estimar costes (para capacidades, control de calidad y programación fijas) con frecuencia comienza desde la concepción del proyecto y continúa aun después de iniciada la codificación.

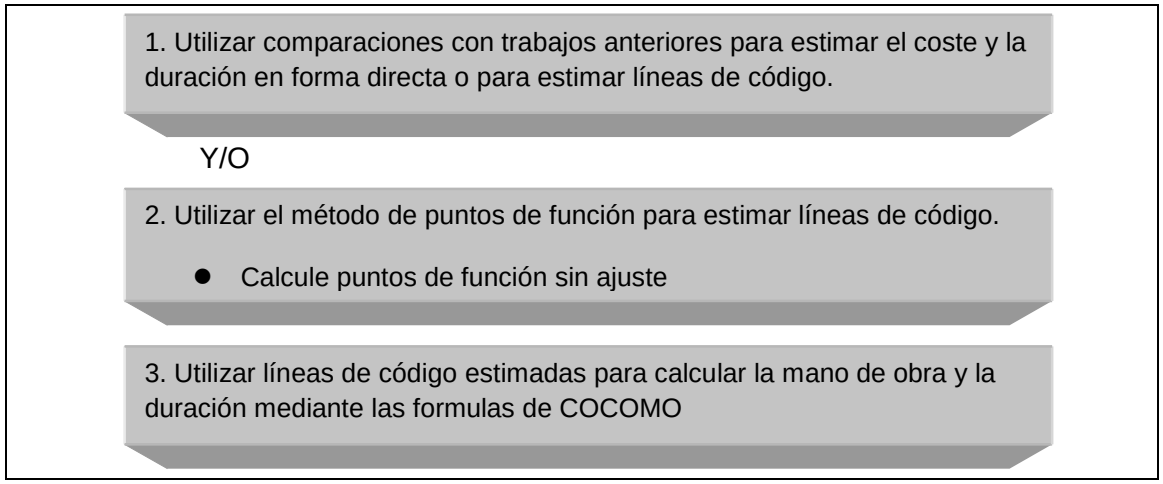


Figura 40.Mapa conceptual para la estimación típica del costo. Adaptado de [Eric J. Braude]

La figura 40, muestra un mapa conceptual típico para las primeras estimaciones de costo y la duración de un proyecto.

Varios métodos de estimación, como el modelo COCOMO, dependen del número de líneas de código (LoC). En las primeras etapas de un proyecto es posible que COCOMO no aparezca muy útil debido a que falta mucho para llegar a la codificación. Sin embargo cuando un producto se puede comparar con otros, es factible estimar las líneas.

Métricas del proceso

Vamos a recordar que el nivel 5 de CMM, requiere mejora continua en el proceso mismo. Aunque pocas organizaciones trabajan en este nivel. A fin de mejorar el proceso de administración de un proyecto se debe preparar para medir su efectividad mediante las métricas de proceso. Estas métricas, que pueden medir la efectividad de la organización del proceso, incluyen la secuencia de pasos. Se mide por separado la efectividad del análisis, diseño, codificación y prueba de los requisitos.

1. Identificación y definición de las métricas que utilizará el equipo en cada etapa; incluidas:
   - Tiempo gastado en: Investigación, ejecución, revisión.
   - Tamaño (como línea de código).
   - Número de defectos por unidad (líneas de código, incluye fuente).
   - Autoevaluación de la calidad en una escala de 1 a 10; siendo 1 la más baja y 10 la más alta
2. Documente esto en el PAQS.
3. Acumule los datos históricos por etapas.
4. Decisión sobre donde se colocarán los datos de medición.
5. Designe ingenieros para administrar la recolección por etapa.
6. Programe revisiones de los datos para aprender de la experiencia

Figura 41.Una manera de reunir métricas de proceso

La figura 41 contiene una secuencia de acciones que se puede tomar durante la vida del proyecto con el fin de mejorar el proceso de manera continua.

4.2.2 Contenido de un SRS

Un SRS no tiene que seguir este contorno o usar los nombres dado aquí para sus partes, un buen SRS debe incluir toda la información que se menciona aquí.

4.2.2.1 Introducción

En esta sección se hace una descripción general de lo que va contener el documento.

Se presenta el análisis global de los requisitos de una aplicación, es un proceso de conceptualización y expresión de los conceptos en forma concreta. La mayor parte de los defectos encontrados en el software entregado se origina durante el análisis de requisitos. En general, estos defectos son los más caros de reparar.

Proporcionar una introducción a todo el documento de Especificación de Requisitos de Software (ERS). Consta de varias sub-secciones: propósito, ámbito del sistema, definiciones, referencias y visión general del documento.

No se intenta establecer de forma rígida como debe ser la especificación de requisitos, trata de establecer una estructura de documento que alcance las características que se describan. El análisis de requisitos permite al analista crear un documento donde se define cual es el ámbito correcto, cuales son los cambios a realizar, que funciones realiza cada unidad del sistema, los recursos necesarios, etc. Este documento es la especificación de los requisitos software y se tiene a partir del estudio del sistema actual y representa las mejoras del nuevo sistema.

Este capítulo constatará de lo siguiente:

- Descripción de la información: Descripción detallada del problema que el software debe resolver. Estarán documentados el flujo y las estructuras de la información, hardware, software, etc.

- Descripción funcional: Proporciona una descripción de cada función requerida para resolver el problema.

- Descripción del comportamiento: Describe el comportamiento del sistema software mediante diagrama de estados, con especificaciones de eventos y acciones, etc.

- Criterios de validación: Incorpora las clases de prueba que se deben realizar al software, los límites de rendimiento y la respuesta esperada.

4.2.2.1.1 Propósito del documento.

Definir el propósito del documento ERS y especificar a quien va dirigido el documento.

El documento de especificación de requisitos es un documento fundamental y es el punto de partida del desarrollo de cualquier sistema software.

El documento de especificación de requisitos, es el encargado de recoger todo el fruto de trabajo realizado durante la etapa de análisis del sistema de una forma integral en un único documento.

El documento de especificación de requisitos debe cubrir todos los objetivos indicados en el análisis de requisitos. Dado que es un documento que deberá ser revisado con frecuencia a lo largo del desarrollo de la aplicación, es fundamental que se redacte de una manera fácil de modificar.

Por otro lado debe facilitar la labor de verificación del cumplimiento de las especificaciones. Todo esto hace que la mejor manera de redactar este documento sea en forma de un contrato con sus distintas cláusulas organizadas y agrupadas según el carácter de los requisitos.

Los requisitos específicos se escriben más que nada para los diseñadores y desarrolladores. Se crean a partir de los requisitos del cliente y de la interacción continua con el cliente. Los requisitos específicos deben ser comprobables y estar clasificados de manera sistemática. Una manera útil para organizar los requisitos específicos es por clases y funciones de dominio. Existen varias métricas contra las que se pueden inspeccionar los requisitos específicos. Un buen análisis de requisitos produce grandes beneficios.

Métricas para el análisis de requisitos específicos

El uso selectivo de métricas maximiza la inversión en la inspección al enfocar el proceso en resultados útiles y cuantificados.

Cada métrica proporciona beneficios, pero su recolección, almacenamiento, análisis y estudio cuestan dinero y tiempo, el arte de aplicar métricas es optimizar la razón costo/beneficio.

Medidas	Descripción
Medidas que evalúan en qué grado están escritos los requisitos	- Porcentaje de requisitos específicos no ambiguos. - Grado en que está completo. - Porcentaje de requisitos específicos mal clasificados. - Porcentajes de requisitos específicos que no son: ● Comprobables ● Trazables ● Con prioridades ● Atómicos ● Consistentes
Medidas de efectividad de la inspección de requisitos	- Porcentaje de requisitos faltantes o defectuosos encontrados por hora de inspección.
Medidas de efectividad del proceso de análisis de requisitos	- Costo por requisito específico ● Costo bruto (tiempo total dedicado/número de requisitos específicos). ● Costo marginal (costo de obtener uno más) ● Tasa a la que los requisitos específicos se: o Modifican o Eliminan o Agregan
Medidas del grado en que un requisito está completo	- Esto se puede estimar después del fin oficial de la recolección de los requisitos específicos, a partir de la tasa a la que los requisitos específicos se: ● Modifican ● Agregan

Tabla 115.Medidas de aseguramiento de la calidad de requisitos específicos

4.2.2.1.2 Alcance

a. *Explicar lo que el software hará, y, si fuera necesario aclararlo, también lo que no se espera que haga.*
b. *Describir para qué se aplicará el software, incluyendo sus beneficios relevantes, objetivos, y metas.*

En esta subsección:

- Se identifica el nombre de los productos software que se van a desarrollar.
- Se explica lo que hacen los productos y lo que no hacen.
- Se describen las aplicaciones del software especificado, incluyendo beneficios y objetivos.

- Se debe ser consistente con especificaciones similares de más alto nivel si existen, es decir, en el caso de que estas sean las especificaciones de un subsistema.

Descripción del producto

La descripción del producto se trata de una parte de la documentación del paquete del producto y proporciona información sobre la documentación del usuario, el programa y los datos del objetivo principal de la descripción del producto para:

- Ayudar a los usuarios o compradores potenciales en la evaluación de la idoneidad del producto para sí mismos.
- Sirva de base para las pruebas.

Requisitos generales

La descripción del producto debe ser lo suficientemente comprensible, completa y fácil para ayudar a los compradores potenciales en la evaluación de la idoneidad para sí mismos, del producto antes de comprarlo.

1. La identificación y las indicaciones

   La descripción del producto deberá tener una identificación única del documento. Puede ser un nombre diferente a partir de la descripción del producto, por ejemplo: descripción de las funciones, la información del producto, ficha del producto.

2. La identificación del producto

   La descripción del producto debe identificar el producto. La identificación del producto deberá tener por lo menos el nombre del producto y una versión o la fecha.

3. Proveedor

   En La descripción del producto deberá figurar el nombre y la dirección de al menos un proveedor.

4. El trabajo de tareas

   La descripción del producto debe identificar el trabajo de tareas con la intención de establecer qué se puede realizar con el producto.

5. Conforme a los documentos de los requisitos

   En la descripción del producto se puede hacer referencia a documentos de requisitos a los que el producto se ajusta, en cuyo caso las ediciones relevantes serán identificadas.

6. Sistema requerido

   El sistema necesario (hardware, software y su configuración) para poner el producto en uso deberá ser especificado, incluyendo el nombre del fabricante y los identificadores de tipo de todas las partes, por ejemplo:

   - Unidad de procesamiento como co-procesadores.
   - Tamaño de la memoria principal.
   - Tipo y tamaño de almacenamiento periférico.
   - Entorno de red.
   - Sistema de software y otros programas.
   - Entrada y salida del equipo.
   - Entre otros.

   Diferentes sistemas requieren que se especifique, por ejemplo, para la tarea de trabajo diferente, diferentes valores límite de requisitos de eficiencia. La declaración (o cualquier otro, si son compatibles...) pueden aparecer en la descripción del producto, si antes un hardware o software ha sido identificado.

7. Interfaz de otros productos

   Si la descripción del producto hace referencia a la interfaz con otros productos, la interfaz o productos se identificarán.

8. Productos para ser entregados

   Todos los componentes físicos de los productos que suministran a los programas deben ser identificados, en particular, todos los documentos impresos y todos los medios de comunicación de datos.
   En los programas de forma suministrada se harán constar, por ejemplo, programas fuente, los módulos de objeto o módulos de almacenamiento.

9. Instalación

   Se indicará si la instalación del producto puede ser llevada a cabo por el usuario.

10. Apoyo

    Se indicará ya sea o no el apoyo para el funcionamiento del producto que se ofrece.

4.2.2.1.3 Definiciones, acrónimos, y abreviaturas

Dar las definiciones de todos los términos, acrónimos (siglas) y abreviaturas que sean pertinentes para el adecuado entendimiento del SRS.

Es un listado de las definiciones de todos los términos, acrónimos y abreviaturas necesarios para interpretar correctamente el SRS. La información puede ser proporcionada con referencias a apéndices. Se debe tener cuidado especial para aclarar términos específicos de la aplicación que estamos utilizando.

4.2.2.1.4 Referencias

a) Especificar las fuentes de las cuales pueden obtenerse los documentos referenciados.
b) Especificar las fuentes de las referencias obtenidas. Esta información puede ofrecerse haciendo alusión a un apéndice o hacia otro documento.
c) Identificar cada documento por título, número de informe, fecha y editorial.

Como parte de documentación utilizada mencionamos la siguiente documentación:

- Descripción del diseño del software (DDS).
- Plan de aseguramiento de la calidad del software (PAQS).
- Plan de documentación del usuario de software (PDUS).
- Documentación de pruebas de software.
- IEEE 1233- 1998 Guía para el desarrollo de especificaciones de requisitos del sistema.

Establecer una gestión y entornos de ingeniería basados en los requisitos de calidad de ISO / IEC 12119: 1994 (E).

4.2.2.2 Descripción global

Se describen los factores generales que afectan al producto y sus necesidades. No se establecen requisitos específicos. En su lugar, proporciona una base para los requisitos que se definen en detalle en la siguiente sección de la especificación de los requisitos del software, y los hace más fáciles de entender.

Se compone de lo siguiente:

- Perspectiva del producto.
- Funciones del producto.
- Características del usuario.

- Restricciones.
- Supuestos y dependencias.
- Distribución de los requisitos.

Factores que afectan al producto y a los requisitos. Esta sección contiene consideraciones generales que afectan a la globalidad del sistema. Proporciona contenidos para entender mejor los requisitos específicos.

4.2.2.2.1 Perspectiva del producto

Se describe si lo que estamos analizando es un producto en su globalidad o es parte de un sistema más amplio. En este segundo caso, describir los demás componentes del sistema. También se suele identificar el hardware a utilizar.

- El formato de representación y el contenido deben ser adecuados al problema. - La información contenida en una especificación se debe anidar. Se debe representar el sistema por niveles donde en cada uno se obtendrá divisiones de las funciones y más detalles de las mismas. Se debe utilizar una numeración sencilla para identificar los niveles. - Se debe restringir el número de diagramas y notaciones. - Las notaciones se deben utilizar de forma consistente. Una misma notación podría representar cosas distintas. Una notación confusa o inconsistente degrada la comprensión y aumenta los errores. - Las especificaciones deben ser revisables. En un entorno adecuado, las herramientas CASE facilitarán el trabajo.

Tabla 116.Representación de los requisitos

En las organizaciones modernas, grandes y complejas, los operarios, los inspectores y la mayoría de los supervisores no tienen un conocimiento completo de la aptitud para el uso del producto, y por ello no está claro donde tienen que poner énfasis ni cómo tienen que tomar decisiones.

- La especificación: Esta es la fuente primaria. Debido a que la especificación raramente refleja información de los clientes y de otras fuentes, es importante que el equipo de trabajo profundice proactivamente en actualizar la información de las diversas fuentes.

- Datos del cliente: la clasificación en sí misma puede ser de gran utilidad en el proceso de toma de decisiones, especialmente si el equipo de trabajo amplía, para la fase de preproducción, la información de partida, incluyendo datos del conjunto de clientes: el comprador, el usuario final, los distintos departamentos involucrados en la producción, la inspección, el embalaje, la expedición y todos los demás. Construida así, la clasificación refleja con mayor precisión las necesidades del cliente y expectativas en el producto.

- Experiencia de fabricación en el producto: Problemas y soluciones inadecuadas, incluyendo la evidencia de no conformidades durante el proceso de fabricación, ayudan a identificar problemas futuros.

- Prueba de duración y ensayo funcional: Un modelo estructurado y formal de clasificación debe incluir funciones del producto cuando estas son parte del proceso de producción. Los resultados del ensayo funcional proporcionan datos para la clasificación. En los casos en los que existe la posibilidad de utilización inadecuada del producto, lo que en general y por razones variadas e imprevistas significa que el cliente utiliza el producto de manera distinta a la de su propósito original, el ensayo puede anticipar y prevenir este tipo de utilizaciones inadecuadas o al menos mitigar sus efectos. La clasificación asegura en este caso que las características del producto afectadas reciben atención adecuada.

- Fallos durante el uso del producto: El fallo en el uso del producto es lo último que se desea. Cualquier característica del producto para la que se puede establecer una relación causal con un fallo en la utilización del producto es, por definición, una característica crítica.

Tabla 117.Clasificación por importancia

La clasificación por importancia es un dato útil, tanto para la planificación de control del proceso como para la planificación general de la calidad. Algunas características de calidad pueden aparecer en distintos niveles, según los criterios de importancia.

Interfaces de sistema

Lo que hay entre los diversos módulos del sistema.

- Enumerar cada interfaz de sistema
- Descripción de la interfaz del software para hacerlo compatible con el sistema.

Lo que hay entre los módulos es flujo de información; por lo tanto se tiene que especificar qué información alimenta a cada uno de los módulos que sean requeridos.

Hay que especificar cómo es esa información (numérica, alfanumérica, rango de valores posibles, unidades de medida, etc.).

Identifica la funcionalidad del software y la descripción de la interfaz. Especifica:

- Características lógicas de cada interfaz entre el producto y sus usuarios.
- Todos los aspectos relativos a la optimización de la interfaz con los usuarios.

Interfaces de usuarios

Lo que estará entre el usuario y el software.

La documentación de requisitos de la interfaz de usuario consiste en documentar y verificar información sobre los usuarios, su trabajo actual y su visión del trabajo con el software futuro, con vistas al diseño de la interfaz de usuario del nuevo software.

- Restricciones organizativas y técnicas. - Perfiles de los usuarios. - Descripción de las tareas actuales y futuras. - Especificación de usabilidad. - Guiones para los casos de uso.

Tabla 118.Documentación de requisitos de la interfaz de usuario

La interfaz de usuario depende en gran medida

1. Comodidad del usuario: Afecta a la ansiedad, frustración, confusión, fatiga. 2. Productividad del usuario: Es mejor en la medida en que sea necesario seleccionar menos teclas y botones y que los recorridos que deba realizar con el ratón sean más cortos y menos frecuentes. 3. Imagen del software: Los usuarios juzgan la calidad del software sobre todo a causa de lo que ven más directamente, es decir, a causa de las interfaces. No se dan cuenta de la calidad de la programación, al menos mientras no afecte de manera perceptible al funcionamiento del software; por lo tanto, que la interfaz de usuario pueda determinar el éxito o el fracaso de un software.

Tabla 119.Factores que intervienen en la interfaz de usuario

Además, una interfaz de usuario inadecuada puede provocar errores del usuario, sobre todo en caso de circunstancias no habituales, en la que el usuario tiene como única guía la interfaz misma. También puede provocar la infrautilización del software (En términos de usuarios potenciales y de funciones disponibles) e incluso su rechazo total.

Elaboración de los perfiles de usuario

No se puede diseñar correctamente una interfaz de usuario sin saber para quién se hace, ya que un diseño apropiado para un usuario (en sentido colectivo) no puede ser para otro. Hay que evitar en especial el error de que los desarrolladores del software diseñen la interfaz de usuario como si los usuarios fueran ellos mismos, ya que tienen una cultura profesional y unos conocimientos de informática muy diferentes a los de la gran mayoría de los usuarios.

- Experiencia en torno hardware y software (ratón, teclado, ventanas etc.).En aplicaciones del mismo dominio. - Experiencia en el trabajo. - Frecuencia de uso del software y grado de rotación del personal

Tabla 120.Perfil de usuario

Documentación de las tareas actuales y futuras

- La tarea en sí, su frecuencia y qué usuarios la realizan. - El entorno en que se lleva a cabo, si hay limitaciones de espacio, la iluminación, si hay suciedad que pueda afectar a algunos dispositivos como los ratones. - Su situación dentro del flujo de tareas, es decir, cuales la preceden, la siguen o la interrumpen y las interdependencias con otras tareas que obligan a seguir un orden determinado. - Que información entra y sale y cuáles son los resultados y hacia dónde van - Que documentos y herramientas son necesarias. - Cuáles son los problemas y errores más frecuentes. - Las quejas y sugerencias sobre cómo se realiza la tarea.

Tabla 121.De cada tarea se recogen los siguientes aspectos

Esta información no se documenta de manera formalizada. Una tarea o secuencia de tareas se puede representar mediante un diagrama de actividades simplificando en el cual sólo se representen los flujos y los estados de actividad, que corresponderían a operaciones, ya sea manuales o informatizadas.

Comparación entre tareas y casos de uso

Las fuentes de información para los requisitos de funcionalidad y de interfaz de usuario son las mismas. Así, al describir los casos de uso desde el punto de vista de la funcionalidad, se pone énfasis en lo que realiza el software, mientras que cuando se describen desde el punto de vista de interfaz de usuario, interesan más las acciones que realiza el usuario y en qué condiciones lo hace.

- Desde el punto de vista de un caso de uso, el software presenta determinada información, desde el punto de vista de las tareas el usuario la comprueba y toma una decisión. - En un caso de uso que tiene diferentes posibilidades, no es necesario indicar la frecuencia de cada uno; en una tarea sí. - A las tareas que son manuales no les corresponde ningún caso de uso, y en un caso de uso tampoco se indican las operaciones manuales de una tarea; en la descripción de una tarea debe tenerse en cuenta, al menos para determinar la duración de la misma. - Cuando se produce una anomalía, el software se limita a avisar al usuario, según el caso de uso, mientras según la descripción de la tarea, el usuario debe tomar una decisión y tal vez empezar otra tarea independiente.

Tabla 122.Diferencias entre tareas y casos de uso

Especificaciones de usabilidad

Las especificaciones de usabilidad son los requisitos no funcionales relativos a la interfaz de usuario.

Las especificaciones de usabilidad se pueden referir a la facilidad de utilización o aprendizaje o a rapidez y precisión en la ejecución de las tareas. Conviene expresarlas de manera cuantitativa.

Interfaces de hardware

Qué hardware usará el software para entrada/salida de información.

Se describen las características de las interfaces entre el producto de software y los componentes de hardware del sistema. Incluye características de configuración, dispositivos que se deben soportar, como deben ser soportados y protocolos. Se especifican las características de cada interfaz entre el producto y los componentes hardware. Incluye la configuración, dispositivos soportados y protocolos.

Una descripción de las características lógicas del hardware tales como: protocolo de las interfaces, control de terminales orientados a pantalla o a línea, etc.

Interfaces de software

Especificar qué otros productos de software se necesitarán (por ejemplo, sistemas manejadores de base de datos, sistemas operativos), interfaces con otras aplicaciones de sistemas, así como los programas para que el software pueda interactuar con los demás módulos.

Se debe especificar el uso de otros productos de software necesarios (sistema de manejo de datos, sistema operativo, librerías o paquetes), Interfaces con otros sistemas de aplicación.

Para cada interfaz se debe indicar: propósito de la interfaz con el producto de software, definición de la interfaz en términos de contenido y formato de la misma.

Una descripción de las interfaces a otro software necesario. Ejemplo: sistemas operativos, paquetes de librería, aplicaciones diversas etc.

Es una lista de otros productos software e interfaces con otras aplicaciones:

1. Para cada producto se debe incluir: Nombre, Mnemónico, Número de versión y Fabricante.
2. Para cada interfaz se debe desarrollar en: Motivos para interactuar con el producto y definición de la interfaz acerca de contenido de los mensajes y formatos.

Interfaces de comunicaciones

Un ejemplo sería un protocolo de comunicaciones para una red de área local.

- Interfaces usadas para utilizar este protocolo.
- Los protocolos de comunicación son normas que deben aportar tales funcionalidades:

- Aceptar la localización de un computador de manera inequívoca. - Aceptar desarrollar una conexión con otro computador. - Aceptar intercambiar información entre computadores de manera segura, independientemente del tipo de equipo que estén conectadas. - Aislar a los usuarios de los enlaces que se utilizan para el intercambio de información. - Aceptar la liberación de la conexión de manera ordenada.

Tabla 123.Normas que aportan funcionalidades en los protocolos que se utilizan en las comunicaciones

Debido a la gran complejidad que conlleva la interconexión de computadores, se han tenido que dividir todos los procesos necesarios para realizar las conexiones en diferentes niveles. Cada nivel se ha creado para dar una solución a un tipo de problema particular dentro de la conexión. Cada nivel tendrá asociado un protocolo, el cual entenderán todas las partes que formen parte de la conexión.

Los protocolos de comunicaciones definen las normas que posibilitan que se establezca una comunicación entre varios equipos o dispositivos, ya que estos equipos pueden ser diferentes entre sí. Una interfaz, sin embargo, es la encargada de la conexión.

Un protocolo de red es una norma o conjunto de normas que establece el método para enviar y recibir datos entre varios computadores. No existe un único protocolo de red, y es posible que en un mismo computador instalado posea muchos protocolos, esto es debido a que el ordenador pertenezca a distintas redes. Esto puede conllevar un riesgo de seguridad, es decir cada

protocolo de red que se instale en un sistema queda abierto a todos los adaptadores de red que existan en el sistema, físicos, lógicos.

Si los dispositivos de red o protocolos no están de la manera correcta configurados, esto conlleva accesos no deseados a nuestros recursos. Por seguridad es conveniente la instalación del número de protocolos indispensables. Hoy en día y en general nos debería bastar con el protocolo TCP/IP.

Una conexión de red implica una relación entre computadores en diversos niveles:

- Se necesita una conexión física.
- Se necesita el manejo de datos transportados.
- Se necesita un sistema de transporte.
- Se necesita mostrar los datos.

Normalmente los protocolos de red trabajan en grupos, encargándose de aspectos parciales de la comunicación.

Operaciones

Especificar los diferentes modos de operación del software por parte del usuario.

Los clientes desarrollan una visión con frecuencia inconsciente e incompleta del modo en que debe operar la aplicación. Usuarios diferentes en general tienen conceptos distintos de lo que implica una aplicación del software. Estos conceptos diferentes de operaciones llevan a aplicaciones muy distintas.

El analista ayuda al cliente a aclarar el concepto de operación. Como los clientes en general no conocen técnicas para expresar los conceptos, los analistas les proponen técnicas adecuadas, como casos de uso, flujo de datos o transición de estados, estos conceptos que hemos mencionado como son: casos de uso, flujo de datos o transición de estados se detallarán en la parte de diseño.

Cuando se utilizan especificaciones de interfaz, no se describe el cálculo que realiza la transformación de entrada en salida. Por el contrario, una especificación operacional define explícitamente esta transformación de entrada y salida, aunque pueda expresarse de una manera abstracta de alto nivel.

Liskov y Berzins dicen que la diferencia entre un lenguaje para los requisitos operacionales y otro de aplicación es que el primero debe diseñarse pensando en la claridad de la expresión. No se deben tener en cuenta consideraciones de aplicación. Sin embargo, lo anterior no excluye que la especificación sea ejecutable; esto tiene ventajas para realizar pruebas y construir prototipos.

De hecho, si el lenguaje de especificación es ejecutable, la especificación no es más que otro problema. Esto no significa, por supuesto, que las especificaciones ejecutables quiten validez a los lenguajes de programación.

Los lenguajes de especificación ejecutables imponen demandas muy fuertes a un sistema a tiempo de ejecución y podrían ejecutarse con una lentitud cientos de veces que los programas equivalentes. La función principal de las especificaciones operacionales sigue siendo la de definir lo que el programa debe hacer en lugar de efectuar realmente las operaciones. Los lenguajes de especificación actuales serán los futuros lenguajes de programación, si se logra diseñar y construir el hardware apropiado a sus necesidades.

Se pueden construir especificaciones de programas utilizando lenguajes realmente sencillos. MCcarthy(1962) mostró cómo podrían escribirse las especificaciones operacionales en un lenguaje que no tuviera proposición de asignación. El único mecanismo de control eran las funciones que podían llamase en forma recursiva, y expresiones que podían ser condiciones.

Requisitos de adaptación a un lugar

Se deben especificar los requisitos de datos o secuencias de inicialización que son específicas a un sitio dado, misión, o modo operacional (por ejemplo, valores posibles, límites de seguridad, etc.); se deben especificar el sitio o las características relacionadas a la misión que deben modificarse para adaptar el software a una instalación particular.

Entorno

El entorno puede influir en forma implícita o explícita, o colocar una restricción en los requisitos del sistema. El analista debe ser consciente de estas influencias en las capacidades del sistema.

En los casos en que los sistemas son sensibles a las influencias ambientales, que los clientes y analista especifiquen las influencias ambientales que afectan a los requisitos del sistema. Las influencias ambientales pueden ser clasificadas en categorías que se superponen, como sigue:

Categoría	Descripción
Políticos	Internacional, federal, estatal y local, agencias gubernamentales tienen leyes y reglamentos que influyen en los requisitos del sistema. Algunas agencias del gobierno pueden tener organizaciones de ejecución que verifican el cumplimiento de sus leyes y reglamentos. Ejemplos de leyes gubernamentales son los derechos de autor, patentes y leyes de marca registrada. Ejemplos de las regulaciones gubernamentales son la zonificación, los riesgos ambientales, residuos, reciclaje, los sistemas de seguridad y salud. Cambios en la influencia política en función de las fronteras políticas. Lo que afecta a los requisitos del sistema en un ambiente puede ser completamente diferente en otro.
Mercado	Condiciones de mercado que influyen en el desarrollo de la especificación del sistema. - Necesidades del cliente a los sistemas mediante el uso de la investigación de mercados o el desarrollo de mercados para que coincida con la investigación técnica. - La demanda se debe considerar, ya que afecta la distribución del sistema y la accesibilidad. La distribución y los requisitos de accesibilidad deben ser identificados durante el desarrollo del sistema y antes de que el sistema está fabricado y/o integrado para permitir que estos requisitos puedan ser incorporados en el sistema. Por lo tanto, es importante tener en cuenta los segmentos de mercado que se dirigen en el sistema y cómo la información de marketing puede ser utilizada en la obtención de los requisitos en el sistema. - La competencia: Conocer los sistemas de la competencia ayudará a definir los requisitos. Mediante los siguientes aspectos: funciones, precio, confiabilidad, durabilidad, rendimiento, capacidad de mantenimiento, sistema de protección y seguridad.
Normas técnicas y políticas	Requisitos del sistema son influenciados directamente por los clientes que tienen que ajustarse a las normas y las políticas técnicas emitidas por el gobierno o la industria. Las políticas, normas técnicas y directrices correspondientes se ajustan a los siguiente aspectos: - Sistema de consistencia: Establecen los requisitos específicos de todos los detalles acerca de cómo un sistema en particular debe ser implementados. - La seguridad del sistema: Normas de seguridad industrial son generalmente impuestas para ayudar a prevenir los riesgos de seguridad y posibles problemas legales. - Sistema de fiabilidad y facilidad de mantenimiento: El cumplimiento de los requisitos de seguridad debe estar claramente identificado en el documento de requisitos del sistema.
Cultural	La cultura es el comportamiento humano, los patrones de integración que se transmiten de generación en generación. Se trata de una experiencia adquirida que se origina a partir de las creencias religiosas, el país de origen, grupo étnico, nivel socioeconómico, idioma, los medios de comunicación, lugar de trabajo, y familia. Para entender la cultura de una región o segmento de mercado, los valores y creencias de las personas deben ser conocidos. La influencia cultural debe ser considerada cuando se desarrolla un sistema, ya

	que afectará a los requisitos del sistema.
Organización	Los requisitos del sistema son influenciados por la organización en la que los requisitos son desarrollados. La Influencia de la empresa en la organización puede tomar la forma de comercialización, la política interna, las políticas técnicas y normas internas. Cada empresa tiene su propia cultura, fines, valores y objetivos que pueden influenciar en el sistema que se desarrolla, fabrica y/o entrega.
Física	Influencias naturales y artificiales, tales como temperatura, radiación, humedad, presión, etc.

Tabla 124.Categorías que influyen en el entorno para adaptar el software a una instalación específica

4.2.2.2.2 Funciones del producto

Es un resumen de las funciones que proporciona el software. A veces esta lista puede copiarse de las especificaciones de alto nivel. Este resumen:

- Debe ser inteligible por el cliente con una sola lectura.
- *Se pueden utilizar elementos textuales o gráficos para mostrar las diferentes funciones y sus relaciones. No es un gráfico del diseño, sólo muestra relaciones lógicas entre funciones y variables.*

1. Panorama del producto.
2. Ambiente de desarrollo, operación y mantenimiento.
3. Interfaces externas y flujo de datos.
4. Requisitos funcionales.
5. Requisitos de operación.
6. Manejo de excepciones.
7. Subconjuntos iniciales y prioridad de instrumentación.
8. Modificaciones y mejoras previas.
9. Criterios de aceptación.
10. Guías y sugerencias de diseño.
11. Índice cruzado.

Tabla 125.Formato de la especificación de los requisitos para la producción del software

Explicación del contenido de la tabla 125

- En los puntos 1 y 2, presenta un panorama de las características del producto y resumen de los ambientes de procesamiento para el desarrollo, operación y mantenimiento del producto, las cuales se definen en la definición del sistema y en el manual de usuario preliminar.

- En el punto 3, especifica las características externas del producto de programación. Incluye despliegues que verá el usuario, formato de informes, un resumen de los comandos de usuario y de las operaciones del informe, diagramas de flujo de datos y diccionario de datos estos deben ser desarrollados a alto nivel.

  Los diagramas de flujo de datos, se pueden representar de manera informal o a través de notación especial, no entramos en detalle a especificar la notación especial ya que durante el transcurso de este documento se explicará con mayor detalle específicamente en el

capítulo de diseño pero hacemos hincapié en algunos de los elementos que lo componen. El origen y destino de los datos se representan por rectángulos y los almacenes de datos por rectángulo sin extremo.

- En el punto 4, estos requisitos se expresan generalmente en notaciones relacionales entre entradas, acciones y salidas.

- En el punto 5, las características de operación tales como tiempo de respuesta de varias actividades, tiempo de procesamiento de varios procesos, número de procesos ejecutados, restricciones de memoria primaria y secundaria, ancho de banda requerido para telecomunicaciones y elementos particulares, como restricciones de seguridad especiales o características de confiabilidad no comunes.

  Las características de operación se deben establecer en términos verificables, así como los métodos para verificarlos, es muy importante que se establezcan en forma rigurosa, de tal manera que se pueda emplear un razonamiento lógico en el establecimiento de requisitos.

- En el punto 6, incluyendo las acciones a tomar y los mensajes a enviar en respuesta, a situaciones no deseadas. Se debe preparar una tabla con las condiciones de excepción y sus respuestas, donde se incluyan todas las categorías posibles, sin incluir el fallo temporal de algún recurso (por ejemplo, fallo de un banco de memoria o del procesador); datos de entrada, valores internos, parámetros incorrectos o fuera de rango, incumplimiento de los límites de capacidad, etc. el conjunto de excepciones identificadas debe ser suficiente para que, si ninguna aparece, el procesamiento proceda de manera normal; en la práctica, sin embargo, no es posible lograr este objetivo.

- En el punto 7, la especificación indica los subconjuntos iniciales y las prioridades de instrumentación para el sistema en desarrollo. Los productos de programación se desarrollan en ocasiones como una serie de versiones sucesivas. La versión inicial puede ser un prototipo esquemático, el cual demuestra funciones básicas del usuario y proporciona un marco de referencia para la evolución del producto.

- En el punto 8, las modificaciones y mejoras previstas se podrán incorporar en el producto después de la entrega inicial.

- En el punto 9, los criterios de aceptación del producto de programación especifican las pruebas funcionales y de rendimiento que se deben realizar, y los estándares que se aplicarán al código fuente, documentación interna, y documentos externos tales como las especificaciones de diseño, plan de pruebas, manual de usuario, principios de operación y procedimientos de instalación y mantenimiento.

- En el punto 10, la especificación de requisitos se refiere a aspectos funcionales y de rendimiento del software, y se hace hincapié en la especificación de las características del producto, sin aclarar cómo se

lograrán estas características. El “como” de la instrumentación del producto es tema del diseño y debe ser diferido hasta esa fase. Durante la planificación y la definición de requisitos se entienden y descubren un conjunto de situaciones. Estas deben ser registradas como sugerencias y guías para los diseñadores del producto y no como requisitos para el diseño del producto.

- En el punto 11, relaciona los requisitos del producto con las fuentes de información empleadas en la definición de requisitos. Conocer las fuentes de requisitos específicos permite la verificación y reexaminación de requisitos, restricciones y suposiciones.

4.2.2.2.3 Características del usuario

Describir sus características a nivel de:

- Nivel educativo.
- Experiencia profesional.
- Capacidades técnicas.

Esta descripción ayuda a comprender los motivos que dan origen a los requisitos técnicos.

- Conocimientos de ordenador. - Experiencia con aplicaciones. - Nivel educativo. - Nivel de lectura. - Aptitudes.

Tabla 126.Nivel de conocimiento y experiencia

- Edad. - Género. - Destreza de mano. - Discapacidad.

Tabla 127.Características físicas de Usuario

- Tipo de utilización de la aplicación. - Frecuencia de utilización. - Tasa de rotación de empleados. - Importancia de la tarea. - Repetitividad de la tarea. - Capacidad prevista. - Categoría del trabajo.

Tabla 128. Características de las tareas y trabajos de usuario

- Aptitud hacia el trabajo. - Motivación. - Estilo cognitivo.

Tabla 129. Características psicológicas del usuario

Conocer al usuario es entender la naturaleza de los usuarios posibles de la aplicación. En las tablas anteriormente mencionadas se describen los factores involucrados. La lista de verificación es una manera de asegurar que se conocen las características básicas de los usuarios previstos. En general, el usuario con menor nivel educativo, capacitación, aptitudes y motivación requiere mayor sencillez, más explicaciones y más ayuda.

Concepto de formación

La formación es algo más que una simple adición de conocimientos y habilidades. Este concepto abarca también las capacidades para actuar, es decir, autonomía de trabajo, disponibilidad, responsabilidad, cooperación, etc.

Además, se incluyen aptitudes y conocimientos específicos para resolver problemas laborales y vitales con éxito.

- Proceso activo de capacidades especializadas y pluriespecializadas.
- Adquisición de conocimientos.
- Habilidades y capacidades para superar las tareas.

Tabla 130.Aspectos que incluye la formación

La variedad de métodos de gestión de la calidad y su aplicación práctica en la industria plantea a muchas empresas un difícil problema. La aplicación irreflexiva y no sistemática de los métodos produce declaraciones de insatisfacción, o incluso de conductas erróneas, llegando a una frustración de los distintos usuarios.

- Nivel 1:
  - Conocer criterios de evaluación.
  - Comprender el sistema de gestión de calidad.
  - Aplicar métodos y estrategias.
- Nivel 2:
  - Conocer y aplicar métodos preventivos de gestión de la calidad.
  - Aplicar métodos estadísticos.
- Nivel 3:
  - Comprender lo relativo a la calidad en el puesto de trabajo.
  - Ejecutar métodos estadísticos.

Tabla 131.Niveles y asignación de tareas

En el área de gestión de calidad se produce una rápida devaluación de la formación básica debido a la rápida velocidad de innovación en nuevas técnicas y métodos. Por ello, cambian las especificaciones sobre el contenido de la formación: el aprendizaje se ha convertido en una tarea cotidiana, que no termina tras probar un examen o adquirir un título, sino que dura toda la vida.

4.2.2.2.4 Restricciones

Descripción general de limitaciones que afectarán a los desarrolladores tales como: políticas, limitaciones de hardware, interfaces con otras aplicaciones, operaciones en paralelo, etc.

- Las restricciones son los requisitos que se imponen a la solución por las circunstancias, la fuerza o la coacción. - las restricciones limitan las opciones abiertas a un diseñador de una solución mediante la imposición de sus límites. - Restricción de seguridad puede ser limitada por la tecnología. - Una lista de restricciones pueden incluir interfaces con los sistemas ya existentes (por ejemplo, el formato, protocolo o contenido) donde la interfaz no se puede cambiar, las limitaciones físicas de tamaño, las leyes de la naturaleza, las leyes de un país en particular, el tiempo disponible y presupuesto, la prioridad (por ejemplo, obligatoria u opcional), o una plataforma de tecnología pre-existente. - Las restricciones pueden aplicarse en todos los requisitos o se especifica en una relación con una capacidad específica o un conjunto de capacidades. - Las restricciones pueden ser identificados de manera independiente a los requisitos (es decir, que no limiten cualquier capacidad específica), o como las limitaciones en las capacidades individuales. Muchas limitaciones, como la elección de la tecnología (por ejemplo, el tipo de sistema operativo), se aplicará a todo el conjunto de capacidades.

Tabla 132.Distintos aspectos en las restricciones

Las restricciones expresan condiciones que deben cumplir el elemento del modelo al cual se asocia. Se representan entre llaves { }, lo cual indica que pueden ser interpretadas por las herramientas CASE. Esta herramienta se explica con mayor detalle más adelante.

En UML hay tres tipos de restricciones relativas a las operaciones:

- Las precondiciones son restricciones que se deben cumplir antes de ejecutar una operación. Su cumplimiento nos garantiza que la operación se ejecuta partiendo de un sistema concreto del sistema. - Las postcondiciones se comprueban al acabar y se comprueba en concreto al acabar la ejecución de una operación, y garantiza que cuando esté terminada la operación, el sistema vuelve a situarse en un estado correcto. - Las invariantes son condiciones que se deben cumplir en todo momento. Se tiene que comprobar al inicio de cualquier operación, excepto si hacemos referencia a orientación a objetos donde hay constructores y al acabar la operación.

Tabla 133.Tipos de restricciones en UML

4.2.2.2.5 Suposiciones y dependencias

Describir aquellos factores que, si cambian, pueden afectar a los requisitos. Por ejemplo, los requisitos pueden presuponer una cierta organización de ciertas unidades de la empresa, o pueden presuponer que el sistema correrá sobre cierto sistema operativo. Si cambian dichos detalles en la organización de la

empresa, o si cambian ciertos detalles técnicos, como el sistema operativo, puede ser necesario revisar y cambiar los requisitos.

Esta sección debe incluir una lista de todos los factores que afectan a los requisitos establecidos. Estos factores no son restricciones de diseño para el software pero si hay cambios en estos factores pueden afectar los requisitos establecidos.
Este punto no quiere indicar restricciones del sistema a desarrollar, solo cuestiones que influyen en la especificación de requisitos. Por ejemplo: sistema operativo, librería numérica, etc.

Si el diseñador y los instrumentadores están al tanto de estos cambios, pueden diseñar y construir el producto de tal manera que los facilite. Las modificaciones previas para el producto pueden ocurrir como resultado de cambios anticipados en el presupuesto o en el objetivo del producto, por adquisiciones de nuevo equipo o como resultado de la experiencia adquirida desde su entrega inicial. En cualquiera de estos casos, es importante conocer e incorporar en la especificación cada uno de los cambios.

En este subproceso, los analistas que trabajan con el cliente identifican la mejor forma de comunicar los requisitos a todas las personas que necesitan entender, revisar, aceptar o usar la SRS. Una sola representación no siempre es adecuada, porque:

- La comunidad de clientes y técnicos por lo general tienen diferentes culturas e idiomas, por lo que los requisitos del sistema mismo puede ser presentados de manera diferente a los técnicos o comunidades de clientes.
- Recuperación de información específica es difícil en algunas representaciones.
- Presentación de las interacciones pueden ser difíciles de hacer en algunos de los métodos de las representaciones.
- De la información en un solo lugar a la información en otro lugar puede ser difícil en algunas representaciones.

Por lo tanto, es importante que los analistas, en colaboración con el cliente, identifiquen la mejor forma de comunicar los requisitos a todas las personas que necesitan entender, revisar, aceptar o usar la SRS. Para lograr esto, las diferentes representaciones deben ser preparadas a partir de la SRS. Estas representaciones no deben ser mantenidas por separado, sino que debe ser derivada de las representaciones, y ser generadas a partir de la SRS.

4.2.2.2.6 Distribución de requisitos

Aquí se dice cuáles son los requisitos que pueden atenderse hasta versiones futuras del sistema.

Lista de requisitos que se dejan para futuras versiones.

Durante algún tiempo se ha discutido quién es el dueño de los requisitos: el cliente o el analista. Para desarrollar esta hipótesis, el análisis de requisitos se divide en dos niveles:

1. El primer nivel documenta los deseos y necesidades del cliente y se expresa en un lenguaje claro para él. Los resultados suelen llamarse requisitos del cliente. La audiencia para los requisitos del cliente es la comunidad del cliente y la secundaria es la comunidad del analista.
2. El segundo nivel documenta los requisitos de manera específica y estructurada. Estos se llaman requisitos del analista. La audiencia primordial de estos es la comunidad del desarrollador y la secundaria la del cliente.

Aunque las audiencias principales para los requisitos del cliente como los requisitos del analista son diferentes, los clientes y analistas trabajan junto para crear productos exitosos. Una manera de asegurar una buena comunicación es hacer que los clientes trabajen junto con los analistas.

Como el análisis de requisitos es crucial para la ingeniería del software. Una línea de investigación, la de especificaciones ejecutables incluye la especificación de requisitos de manera que se puedan traducir en forma automática en un código ejecutable. Dado que el estilo de los requisitos es declarativo, es decir, enunciado de hechos; las especificaciones ejecutables necesitan un proceso que convierte los enunciados declarativos en comandos del ordenador.

Durante mucho tiempo se pensó que las herramientas CASE podrían simplificar el proceso de dar seguimiento a los requisitos. El uso de tales herramientas ha pasado por ciclos de optimismo y pesimismo.

El proporcionar una biblioteca de patrones orientados a objetos de alto nivel contra los cuales el analista intenta hacer coincidir el proyecto. Muchas personas piensan que muchos requisitos se pueden construir a partir de bibliotecas de casos de uso existentes.

4.2.2.3 Organización de los requisitos específicos

Indicar los requisitos a un nivel de detalle suficiente que permita a los diseñadores diseñar el sistema que satisfaga estos requisitos, y que permita al equipo de pruebas planificar y realizar las pruebas que demuestren si el sistema satisface, o no, los requisitos. Todo requisito aquí especificado describirá comportamientos externos del sistema, perceptibles por parte de los usuarios, operadores y otros sistemas.

En la especificación de los requisitos del software sirve para documentar un acuerdo entre cliente y equipo técnico teniendo en cuenta las siguientes definiciones:

Nombre	Definición
Cliente	El cliente es un término colectivo que puede incluir al cliente propuesto en el sistema, el organismo económico, aceptación del cierre mediante la sección de entrega, y los directivos que serán responsables de supervisar la implementación, operación y mantenimiento del sistema. Todos los clientes que tienen intereses y preocupaciones deben quedar plasmados en la SRS. Además, algunos clientes no pueden entender el proceso de establecer los requisitos o el proceso de creación de un sistema. Aunque el cliente es el responsable de la aplicación que define al sistema, por lo general, puede no estar familiarizado con el vocabulario y las técnicas de representación que se utilizan a menudo para especificar los requisitos. Dado que uno de los principales objetivos del sistema de análisis de requisitos es asegurar que la SRS se entiende, será necesario ofrecer a los clientes una representación de la SRS en un idioma que el cliente entiende y que esté completa, concisa y clara.
Equipo técnico	La SRS debe comunicar las necesidades del cliente al equipo técnico. La comunidad técnica incluye los analistas, los estimadores, diseñadores, agentes de control de calidad, certificadores, desarrolladores, ingenieros, integradores, probadores, desarrolladores, y los fabricantes. Para estas personas la representación de la SRS debe ser técnicamente precisa y presentada en un formalismo de la que se puede diseñar, construir y probar el sistema requerido.

Tabla 134.Definiciones entre cliente y equipo técnico

Los requisitos específicos son una lista completa de las propiedades específicas y la funcionalidad que debe tener una aplicación, expresada con todo detalle. Cada uno de estos requisitos se enumera, etiqueta y se registra su traza durante toda la implementación. Son consistentes con, y son un refinamiento de, los requisitos del cliente. En esencia se supone que los desarrolladores leerán los requisitos específicos. Los clientes también están interesados en ellos y casi siempre pueden comprender y comentar muchos de ellos.

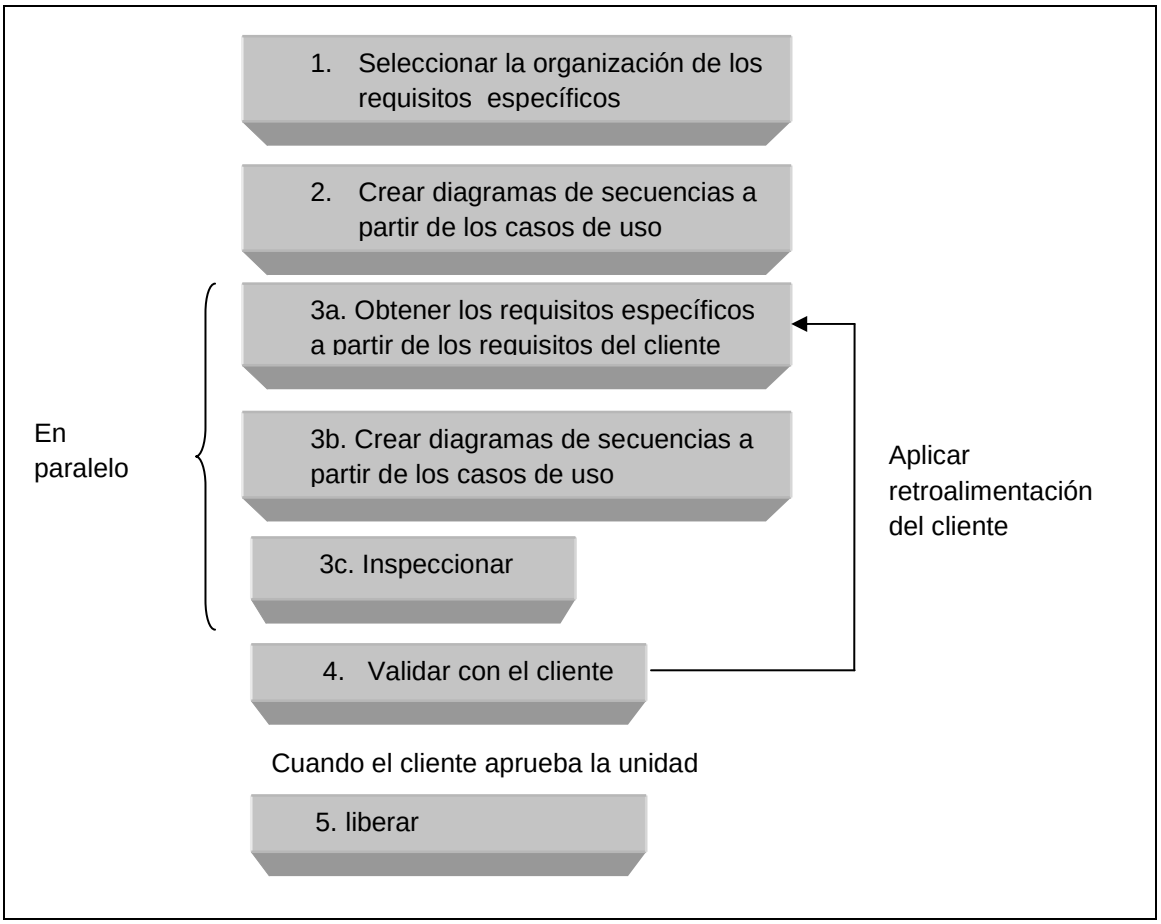


Figura 42.Organización conceptual para los requisitos específicos. Adaptado de [Eric J. Braude]

En la figura 42, se muestra una secuencia común de actividades para reunir y documentar los requisitos específicos. En la primera actividad (1.) describe la forma en que se pueden organizar los requisitos específicos. Los requisitos específicos se escriben a partir de los requisitos del cliente. Es fundamental empezar a escribir las pruebas de cada uno de los requisitos al mismo tiempo que estos. Por consiguiente los requisitos específicos están escritos fundamentalmente para los desarrolladores, tanto los requisitos específicos como sus pruebas se revisan con el cliente.

4.2.2.3.1 Interfaces externas

Descripción detallada de todas las entradas y salidas del sistema del software.

Especifican las características externas del producto de programación. Incluye despliegues que verá el usuario formatos de reportes, un resumen de los comandos del usuario y de las opciones de reportes, diagramas de flujo de datos y diccionario de datos. Las especificaciones para despliegues y reportes son refinamientos de la información contenida en la definición del sistema y en el manual de usuario preliminar.

La visión más abstracta de un componente de software es considerado como una caja negra. Los únicos aspectos exteriores visibles de una caja negra son sus entradas y salidas, de modo que los requisitos de interfaces implican especificar las restricciones de entrada y salida que definen la función del componente de software.

Uno de los aspectos fundamentales de la especificación de requisitos es la interfaces externas del software. La influencia en la sencillez de utilización del software, es decir, lo que más fácilmente percibe el usuario y donde más influyen sus preferencias. En el caso de las entradas podemos encontrarnos con teclados, pantallas de introducción de datos, sensores, ficheros. En el caso de salidas, podemos hablar de pantallas de presentación de información, listados o salidas en papel, ficheros.

A partir del trabajo realizado por Hoare(1969) en este campo: el trabajo de Hoare trataba de la verificación de programas, lo que le llevó a una expresión natural de las especificaciones mediante conceptos matemáticos, como los conjuntos, cuyas propiedades son bien conocidas.

Una ilustración sencilla de la utilización de los requisitos de interfaces, considera la especificación de un procedimiento MAX, cuya función es determinar el mayor miembro de un conjunto de enteros.

```
MAX(X: conjunto de enteros)-> Enteros
Entrada no está vacío(X)
Salida miembro(X,MAX) y
Forall E, miembro(X,E) : MAX >=E
```

Figura 43.Especificación de interfaces, especificación de una función MAX

En la figura 43, se procede a explicar con detalle su funcionalidad:

- La primera línea de esta especificación establece el nombre de la función, MAX y su parámetro de entrada formal, X cuyo tipo es conjunto de enteros, indica que la función devuelve un valor entero, al que hace referencia mediante de función MAX.
- La segunda línea establece la restricción de entrada de que X no debe ser el conjunto vacío.
- Las líneas tercera y cuarta definen las restricciones de salida.
- La primera restricción es que MAX debe ser un miembro del conjunto X
- La segunda restricción especifica que MAX debe ser mayor o igual que todos los demás miembros de dicho conjunto.
- En la especificación de MAX, la entrada se restringió a tener al menos un miembro del conjunto; el conjunto no se aceptó. Sin embargo, no se indicó en absoluto la acción de tomar si no se cumple esta restricción de entrada.
- En la práctica, los componentes del software suelen tener que utilizar entradas incorrectas, por lo que hay que tener un mecanismo para especificar las salidas que corresponden a las entradas correctas o incorrectas. Esto se puede solventar considerando el estado del componente al terminar.
    - Si se proporcionó una entrada válida, se dice que el componente terminó en un estado normal.
    - Si se proporcionó una entrada inválida, se dice que el componente terminó en un estado de excepción.

Tabla 135.Explicación figura 43

4.2.2.3.2 Funciones

Los requisitos funcionales deben definir las acciones fundamentales que deben tener lugar en el software, aceptando y procesando las entradas, procesando y generando las salidas.

En general, la mayoría de los documentos de requisitos del software comienzan con una definición de los requisitos funcionales del sistema. Los analistas de sistemas algunas veces usan el término "especificación funcional" para esta parte de la definición de requisitos. Los requisitos funcionales del sistema son aquellos servicios que el usuario espera del sistema. En general, al usuario no le interesa como se aplican esos servicios, así que el ingeniero del software debe evitar la inclusión de conceptos de aplicación.

En principio, los requisitos funcionales de un sistema deben ser completos y consistentes.

- Completos: Todos los servicios requeridos por el usuario deben especificarse.
- Consistencia: Significa que ninguna definición de requisitos debe contradecir a otra.

En la práctica, y para sistemas grandes y complejos, es casi imposible lograr que los requisitos sean consistentes y completos en la versión inicial del documento. A medida que se descubren los problemas durante las revisiones o en las etapas posteriores del ciclo de vida, el documento debe modificarse en consecuencia.

1. En lenguaje natural: Por la simple razón de que es el más expresivo y porque puede ser comprendido tanto por los usuarios como los desarrolladores del sistema. 2. En un lenguaje estructurado o en un formato que tenga algunas reglas pero no una especificación sintáctica o semántica rigurosa: Los lenguajes estructurados tienen la ventaja debido a las reglas que gobiernan su uso, pueden construirse dispositivos de software para ver si son consistentes y completos. 3. En un lenguaje formal de especificación con una sintaxis y semántica rigurosa definidas: El desarrollo de métodos de especificación formal se ha utilizado de manera muy limitada en la ingeniería de software industrial.

Tabla 136.Maneras de expresar los requisitos funcionales de un sistema.

Los requisitos funcionales son los destinados a establecer el modelo de funcionamiento del sistema y serán el fruto fundamental de las discusiones entre el analista y el cliente. Las personas no muy expertas en sistemas software, tales como el usuario, tienen tendencia a creer que los requisitos funcionales son los únicos a tener en cuenta y que solo en base a ellos se realizarán las pruebas de verificación del sistema después de su implementación.

Sin embargo existen las restricciones o requisitos no funcionales y que están destinados a encuadrar el sistema dentro de un entorno de trabajo.

- Capacidades mínima y máxima. - Interfaces con otros sistemas. - Recursos que se necesitan. - Aspectos de seguridad. - Aspectos de fiabilidad. - Aspectos de mantenimiento. - Aspectos de calidad. - ….etc.

Tabla 137.Delimitación de los requisitos no funcionales

Estos requisitos no funcionales tienen un origen más técnico y no tienen tanto interés para el cliente como lo tienen los requisitos funcionales. Por ende deben permanecer claramente separados en el modelo del sistema.

- Un requisito no funcional de un sistema es una restricción u obligación impuesta al servicio de éste. - Se ven afectados por los cambios en la tecnología de hardware. Puesto que el tiempo de desarrollo de un gran sistema puede ser de varios años, es probable que el hardware disponible al concluir el proyecto sea más potente que el disponible cuando se concibió el proyecto. - Los requisitos no funcionales dependientes del hardware pueden especificarse de modo que se presupongan las capacidades de hardware que existirán a la terminación del proyecto. - Los requisitos no funcionales son tales que tienden a estar en conflicto y actuar recíprocamente con otros requisitos funcionales del sistema. El conflicto entre los requisitos de velocidad de ejecución y los de memoria son un ejemplo. - Los requisitos no funcionales presentan requisitos de manera que los conflictos sean claros y se pueda discernir entre los posibles intercambios.

Tabla 138.Definición de requisitos no funcionales

Boehm(1974) describió una notación llamada matriz de requisitos/propiedades es útil para determinar los intercambios que se deben hacer al tener en cuenta los requisitos. Identifica propiedades de los programas como velocidad de ejecución, necesidades de memoria, confiabilidad, facilidad de mantenimiento, etc., requisitos definidos, tiempo de respuesta al usuario.

Prop. / Req.	Velocidad de ejecución	Facilidad de mantenimiento	Memoria	Confiabilidad	……
Conjunto de caracteres estándar	O	D32	O	D42	
Tiempo de respuesta	D13, D54	B	A	A	
……					

Tabla 139.Matriz de requisitos (Req.)/propiedades (Prop.). Adaptado de [Ian Sommerville]

La matriz se llena de la siguiente manera:

O: Irrelevante.
A: Analizado.
B: En proceso de análisis.
D_i: Supuesto por la definición D_i.
R_j: Se superpone al requisito R_j.

El uso de esta matriz relaciona cada requisito/propiedad con una definición explícita. Reduce la probabilidad de que se incluyan requisitos no realistas al asegurar que se han hecho un análisis completo de las implicaciones de ese requisito. Aunque la matriz no muestra los resultados del análisis, registra si se ha hecho o no un análisis. Es poco probable que se ignoren las implicaciones de un requisito determinado.

- Obligaciones impuestas a los tiempos de respuesta del sistema.
- Limitaciones en la cantidad de memoria que ocupará el software.
- Representación de los datos del sistema.

Tabla 140.Ejemplo de requisitos no funcionales

Tipo de requisito	Descripción
Requisitos del producto	Especifica el comportamiento del producto, Ejemplo: - Requisitos de rendimiento en la rapidez de ejecución del sistema y cuanta memoria se necesita. - Requisitos de fiabilidad: Tasa de fallos para que el sistema sea aceptable. - Requisitos de portabilidad - Requisitos de usabilidad.
Requisitos organizacionales	Se derivan de políticas y procedimientos existentes en la organización del cliente y en la del desarrollador. Ejemplo: - Normas que se deben emplear en los procesos. - Requisitos de implementación: Como lenguajes de programación o el método de diseño a utilizar. - Requisitos de entrega: especifica cuando se entregará el producto y su documentación
Requisitos externos	Son todos los requisitos que se derivan de los factores externos al sistema y de su proceso de desarrollo, incluyen: - Requisitos de interoperabilidad: Se define cómo el sistema interactúa con sistemas de otras organizaciones. - Requisitos legislativos: Debe asegurarse que el sistema funcione dentro de la ley - Requisitos éticos : Deben

	asegurar que será aceptado por el usuario y por el público en general

Tabla 141.Tipos de requisitos no funcionales

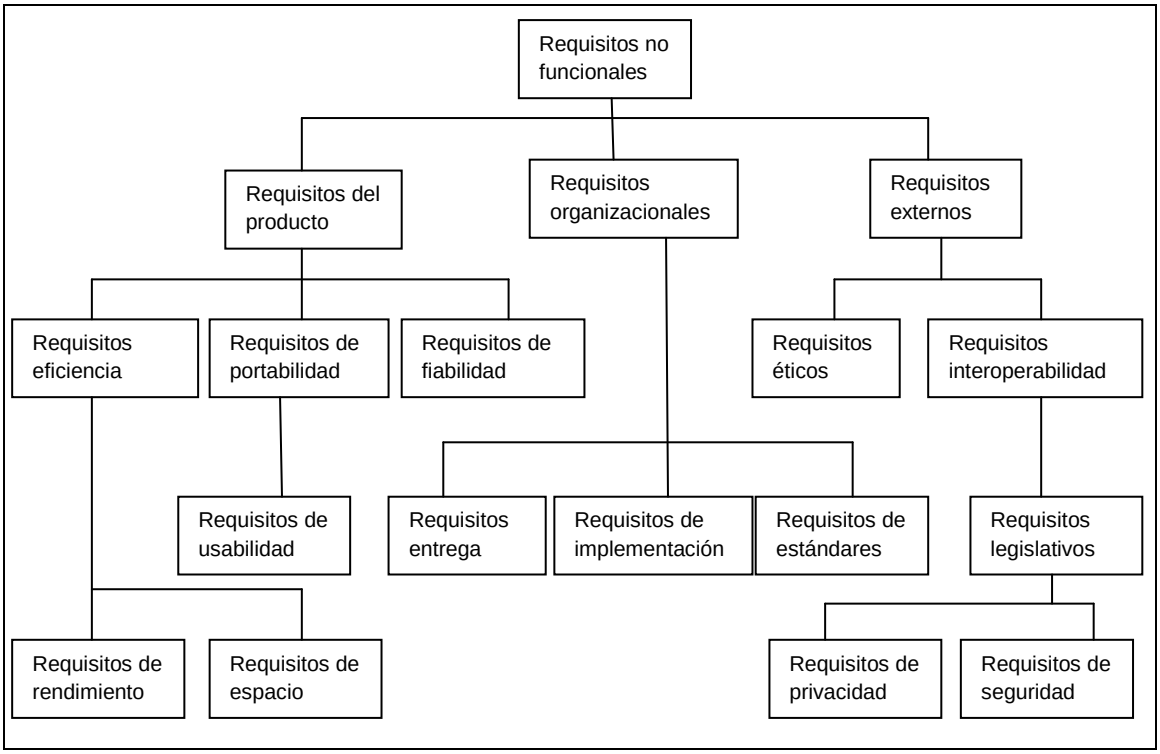


Figura 44.Esquema de los tipos de requisitos no funcionales. Adaptado de [Ian Sommerville]

4.2.2.3.3 Requisitos del desarrollo

Detallar los requisitos (estáticos y dinámicos) relacionados con el almacenamiento que se espera tenga que soportar el sistema.

Definición de un requisito bien formado

Como se definió anteriormente, un requisito bien formado es una declaración de la funcionalidad del sistema (la capacidad) que pueden ser validados, que se deben cumplir o en posesión de un sistema para resolver un problema de un cliente o para alcanzar un objetivo de los clientes, y que está calificado por condiciones medibles y limitada por las restricciones.

Esta definición ayuda en la clasificación de las necesidades del cliente en general. Los requisitos pueden ser tomados de las necesidades del cliente y puede ser derivado del análisis técnico. La definición proporciona un medio para distinguir entre las necesidades como las capacidades y los atributos de los requisitos (condiciones y restricciones). Las restricciones pueden ser funcionales o no funcionales.

Los requisitos de rendimiento abarcan tanto requisitos estáticos como dinámicos.

A veces se identifican los requisitos estáticos bajo una sección separada titulada la Capacidad. **Las capacidades** Son los requisitos fundamentales del sistema y representan las características o funciones del sistema necesarios o deseados por el cliente. Una capacidad por lo general debe indicarse de tal manera que se describe lo que debe hacer el sistema. La capacidad también debe afirmarse de una manera que es una solución independiente. Esto permitirá considerar diferentes maneras de responder a la necesidad o de proporcionar la característica o función. Las condiciones se pueden medir mediante atributos cualitativos o cuantitativos y las características que se estipulan para la capacidad. Asimismo, califican la capacidad que se necesita, y proporcionan los atributos que permiten una capacidad de formularse y expresarse de manera que pueda ser validada y verificada. Las condiciones pueden limitar las opciones abiertas a un diseñador. Es importante identificar las condiciones como atributos de las capacidades, no como las capacidades primarias, para asegurar que los requisitos se definen claramente, sin imponer límites innecesarios en el espacio de soluciones.

Tabla 142.Requisitos estáticos

Se refieren al rendimiento de operación del sistema: Con cuanta frecuencia son realizadas ciertas funciones del sistema, cuanto de rápida debe ser la respuesta del sistema, etc. Es muy importante que estos requisitos se describan de forma cuantificada. Estos requisitos deben especificarse en unidades medibles.

Tabla 143.Requisitos dinámicos

¿Por qué es necesaria la medición? Hay varias razones por las que es necesaria la medición del rendimiento y por qué deberá hacerse un planteamiento organizado.

- Las medidas del rendimiento indican el grado de cumplimiento de los objetivos y, por tanto, cuantifican el progreso hacia la consecución de las metas.
- Las medidas del rendimiento son necesarias para controlar el proceso de mejora continua, que es capital para los cambios requeridos para ser competitivo.
- Las medidas de rendimiento individual, del equipo y de la unidad del negocio se requieren para las revisiones periódicas del rendimiento a cargo de la dirección.

Una vez fijadas las metas y descompuestas en submetas, se necesita establecer medidas clave, es decir, indicadores de rendimiento. Un sistema de medición que controle claramente el rendimiento frente a los planes tiene las siguientes propiedades:

- Indicadores que ligan estrechamente las metas estratégicas con la visión y misión de la organización.

- Indicadores que incluyen los intereses de los clientes; es decir, las medidas que enfocan a las necesidades y requisitos internos y externos.
- Un pequeño número de medidas clave de los procesos clave pueden obtenerse fácilmente en el momento necesario para la toma ejecutiva de decisiones.
- Identificación del desperdicio crónico o coste de la baja calidad.

4.2.2.3.4 Requisitos de la base de datos lógicos

Especificar los requisitos lógicos para cualquier información que será almacenada en la base de datos. Esto puede incluir lo siguiente:

a) Tipos de información usada por varias funciones.
b) Frecuencia de uso.
c) Capacidades de acceso.
d) Entidades de los datos y sus relaciones.
e) Restricciones de integridad.

Requisitos lógicos que tiene que tener la información que se deje en la base de datos. Por ejemplo: Tipos de información, frecuencia de uso, etc.

Muchos grandes sistemas de software necesitan una base de datos de información. Conforme se ejecuta, el sistema toma información de la base de datos y se la proporciona. En algunos casos, la base de datos es independiente del sistema operativo; en otros, se crea para el sistema en desarrollo. De cualquier manera, se necesita una definición de la forma lógica de esta base de datos.

Una técnica que se ha empleado para definir la forma lógica de una base de datos es utilizar un modelo relacional de datos como el descrito por Codd (1970). Al utilizar el modelo relacional, la estructura lógica de datos se especifica como un conjunto de tablas, algunas de las cuales tienen claves comunes.

Este modelo permite definir las relaciones entre los datos elementales sin considerar la organización física de la base de datos. Corresponde al diseñador estudiar la definición de los datos y determinar la manera en que se puede realizar mejor esa especificación considerada como una estructura de datos física. Si se dispone de un sistema administrador de bases de datos, puede incluso aplicarse directamente la definición lógica. Aunque una definición relacional de los datos no compromete la independencia de estos, contiene implicaciones de cómo se van a estructurar físicamente los datos.

Una técnica alternativa para la definición de datos descrita por Chen (1976), utiliza un modelo relacional modificado donde las relaciones no son más que relaciones binarias. Las relaciones binarias se emplean en el lenguaje de definición de requisitos ya analizado.

- El problema de la definición relacional binaria de una base de datos es que a menudo resulta menos intuitiva y más difícil de formular y leer que una definición relacional. - Es responsabilidad de la administración decidir si la mayoría de estas técnicas compensan sus dificultades.

Tabla 144.Inconveniente de la relación binaria

Se ha hecho la existencia de tipos de datos particulares como arreglos y conjuntos, junto con sus operaciones asociadas. Sin embargo, para que las especificaciones sean completas, es importante definir el significado del tipo de datos mediante la especificación del comportamiento de las operaciones de los tipos.

Cuando se representa un tipo de dato como un conjunto de operaciones y valores, se denomina tipos de datos abstractos. Esta es una de las nociones más importantes que se han producido en la ciencia de la computación. El uso de tipos de datos abstractos permite construir especificaciones estructuradas y también conduce a una metodología de aplicaciones que da como resultado un software más fácil de mantener.

4.2.2.3.5 Restricciones del diseño

Especificar las restricciones del diseño que pueden imponerse por otros estándares, las limitaciones del hardware, etc.

Las restricciones de diseño describen los límites o condiciones para diseñar la aplicación. Estos requisitos no pretenden sustituir el proceso de diseño, solo especifican las condiciones que el cliente impone en el proyecto, el entorno u otras circunstancias.

Con frecuencia se imponen restricciones sobre las herramientas y lenguajes. Estas restricciones incluyen históricos dentro de la organización, compatibilidad y experiencia del programador.

Las restricciones de diseño se imponen al proyecto porque los interesados las requieren. Tales restricciones limitan la libertad de diseño de los desarrolladores.

Como parte de la preparación de un diseño de alto nivel, el equipo de diseño debe abordar los criterios explícitos que se usarán para evaluar los diseños y características del diseño alternativos.

- Ajustarse a las necesidades de los clientes. - Ajuntarse a las necesidades de proveedores y productores. - Igualar o superar la competencia. - Optimizar los costes combinados de clientes y proveedores. - El impacto de cada característica sobre las necesidades. - Viabilidad y riesgos de las características propuestas. - Impacto sobre el coste del producto. - Requisitos de normas, regulaciones, políticas, obligaciones, etc.

Tabla 145.Criterios que deben cumplir todos los diseños

Aceptación de las normas

Esta subdivisión debe especificar los requisitos derivados de estándares existentes o regulaciones.

Muchas veces las políticas de la compañía o de los clientes determinan las restricciones de seguir en ciertos estándares.

Es también asegurarse de que se han identificado y cumplimentado todas las normas, regulaciones y políticas relevantes. Algunos de estos requisitos son sólo pautas de cómo debe funcionar un producto o una característica de producto en particular; otros imponen la forma de funcionamiento. Unas pueden provenir de dentro de la organización y otras de gobiernos federales, estatales o locales de agencias reguladoras o de asociaciones sectoriales. Se deben analizar todas las características del producto y las metas de las características del producto frente a estos requisitos, antes de hacer la selección final de las características del producto que se incluirá en el diseño.

Es importante que si existe conflicto al evaluar las características del producto frente a cualquier norma, regulación o políticas. Algunas veces se puede trabajar para recibir la aceptación de un cambio que cumplirá con las necesidades del cliente. Esto ocurre sobre todo cuando se trata de políticas internas.

4.2.2.3.6 Atributos del software del sistema

Hay varios atributos del software que pueden servir como requisitos. Es importante que los atributos se especifiquen para que su logro pueda verificarse objetivamente.

Son propiedades del sistema. El hecho de tenerlas constituye un requisito.
Se presta especial atención a los aspectos de calidad. Los requisitos deben ser medibles y verificables.

Se trata de medir características del software que dependen de la visión externa e interna del producto. Uno de los objetivos es la obtención de producto de buena calidad. La calidad del software es, seguramente el atributo externo de los productos software que más interesa medir. El mayor problema de la

medición de la calidad del software es la complejidad del concepto que se maneja, y por tanto la dificultad en definirlo.

Fiabilidad

Especificar los factores requeridos para establecer la fiabilidad requerida del sistema del software al momento de la entrega.

Definición. La norma ISO / IEC 9126: 2004 define la "fiabilidad" como: "Un conjunto de atributos que influyen en la capacidad de software para mantener su nivel de desempeño bajo las condiciones establecidas para un período determinado de tiempo. "

Las definiciones de sub-características: madurez, tolerancia a fallos y recuperación.

A pesar de que la fiabilidad es a menudo tomada como ejemplo para la definición de los objetivos de calidad cuantitativos, en la práctica es bastante difícil de definir. En particular, las nociones de "nivel de rendimiento", "declaró condiciones" y "fracasos" son muy difíciles de definir. Es por eso que los requisitos cualitativos, como se propone a continuación, son a menudo la única forma práctica de adquirir confianza en la fiabilidad de los sistemas de software.

Además, las técnicas de modelización cuantitativa de fiabilidad que se basan en las pruebas estadísticas no son prácticas de usar, y la definición de un perfil de productos operativos pertinentes es muy difícil en la mayoría de las situaciones.

- **Requisitos del proceso**

  El proceso de desarrollo de software es un dominio donde los requisitos de fiabilidad son más importantes. Estos requisitos pueden solicitase en las distintas etapas del desarrollo.

  - En el nivel de especificación, donde una gran cantidad de fallos tienen su origen, la entidad adquirente puede requerir enfoques o métodos de verificación fuerte. Uno puede imaginar los métodos formales o, al menos, los métodos semi-formales. Lo importante es que las interfaces y el comportamiento del software se ajusten a los ambientes previstos.

  - Para el software crítico, técnicas como el "modo de Fallo, efectos y análisis de criticidad" pueden ser útiles. Permiten identificar los elementos del software que pueden provocar fallos, hay que tener especial cuidado con estos elementos y demostrar la presencia de mecanismos de recuperación.

  - Requerir el uso de lenguajes de programación específicos junto con las reglas de programación estricta. En particular, algunos lenguajes ofrecen construcciones que, aunque de gran alcance, pueden

conducir a errores que son muy difíciles de detectar. Por lo tanto, si los lenguajes son usados, la restricción de ellos es un buen requisito.

- Para las fases de prueba, la entidad adquirente puede exigir que los aspectos específicos de la fiabilidad de la prueba se traten con un cuidado especial. Se puede exigir, por ejemplo, cada modo de fallo identificado y los mecanismos de protección correspondientes a las pruebas durante las pruebas de integración.

- Los requisitos de seguimiento del proyecto. Para mayor fiabilidad, los requisitos cualitativos se expresan a menudo en el proceso de desarrollo. Por lo tanto, es necesario tener confianza en la buena aplicación de este proceso. Esta confianza puede obtenerse mediante la "fiabilidad" de actividades relacionadas con el proceso. Para ello, la entidad adquirente podrá:

  - Participar en la validación del modelo de pliego de condiciones.
  - Indicadores de seguimiento de la aplicación de las recomendaciones resultantes de "modos de fallo, efectos y análisis de criticidad".
  - Vigilar los indicadores de seguimiento de los defectos detectados durante las fases de prueba.

Figura 45.Los principales requisitos que se puedan expresar para satisfacer los objetivos de fiabilidad

- **Los requisitos de aceptación**

  El requisito principal debe tener la aceptación de los informes de justificación resultantes de las actividades específicas destinadas a asegurar la fiabilidad del software. Dado que la mayoría de estas actividades se llevan a cabo con el fin de aumentar la confianza, es importante para obtener una prueba documental de su realización. Los requisitos se pueden orientar a la forma y el contenido de los informes de justificación. Pueden ser derivados del método que ha sido necesario.

  Otro requisito para la aceptación de los objetivos de fiabilidad puede estar basado en el modelado de fiabilidad. Esto se relaciona con el cuadro negro de prueba y medición de tiempo medio entre fallos. Sin embargo, este método sólo es aplicable si el perfil de uso del software se puede definir bien.

Se dice que un software es fiable cuando el tiempo medio transcurrido entre dos errores es máximo.

El objetivo más habitual del estudio y la medición de la fiabilidad del software es resolver el problema de predecir cuándo podría fallar un sistema.

Uno de los supuestos que sirve de base de trabajo en el estudio de la fiabilidad consiste en asumir que los fallos ocurren probabilísticamente en el tiempo según una tasa de intensidad de fallos. En función de los datos de fallos, se construye un modelo que permitan predecir la evolución de la fiabilidad del software en el futuro. Una de las principales clases de modelos de fiabilidad se

basa en considerar que el número de fallos observados en un intervalo de tiempo (0, t) está generado por un proceso Poisson.

- Un modelo de proceso Poisson homogéneo (modelo de intensidad de fallos constante) permite caracterizar el comportamiento de los fallos de un programa en la fase de operación y entre distintas versiones, en el caso de ausencia de depuración y corrección del software.
- Un modelo de proceso Poisson no homogéneo con una función de intensidad de fallos decreciente (modelo de fiabilidad creciente) es aplicable cuando se efectúan correcciones en el software en el momento en que se observa un fallo, por ejemplo, en la prueba del sistema.

- El tiempo medio hasta el siguiente fallo (MTTF: Tiempo medio entre fallos)
- El tiempo medio de reparación del software(MTTR: Tiempo medio de reparación) está como tiempo medio entre fallos tanto MTTF como MTBF
- Tiempo medio entre fallos (MTBF: Tiempo medio entre fallos), se calcula de la siguiente manera:

$$MTBF = MTTF + MTTR$$

- La disponibilidad(D) de un sistema o probabilidad de que un componente está operando en un determinado momento se puede concebir de la siguiente manera:

$$D = \left({}^{MTTF}/_{MTBF}\right) \times 100\%$$

- La densidad de fallos se suele concebir como una función:

$$\left(pdf(t)\right)$$

Esta función se obtiene a partir de datos de ejecución (por ejemplo, de pruebas) y describe la incertidumbre sobre cuando fallará un componente de software.

Figura 46.Métricas empleadas para describir el comportamiento del software en cuanto a fiabilidad

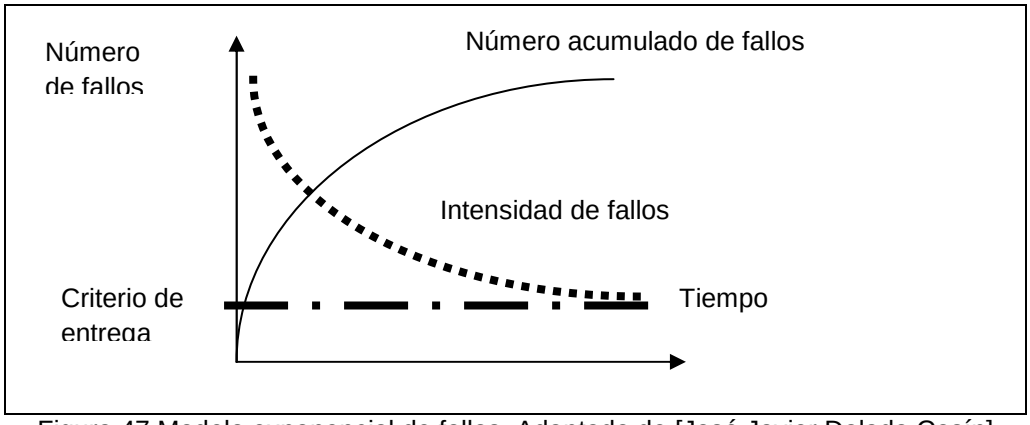


Figura 47.Modelo exponencial de fallos. Adaptado de [José Javier Dolado Cosín]

Uno de los beneficios que supone encontrar un modelo de fiabilidad adecuado radica en que permite ahorrar tiempo en las pruebas del software y estimar el grado de fiabilidad del producto, ya que se puede especificar en función de una cierta intensidad de fallo que se encuentra en la figura 47.

Usabilidad

Definición. La norma ISO / IEC 9126:2004 define la "usabilidad" como: "Un conjunto de atributos que influyen en el esfuerzo necesario para su uso, y en la evaluación individual de tal uso, implícita por un conjunto de usuarios.

Se proponen, además, las definiciones de sub-características: comprensibilidad, facilidad de aprendizaje y operatividad.

Los objetivos de la usabilidad no son fáciles de precisar. Un aspecto clave es la identificación de las categorías de usuarios. En particular, el promedio de conocimiento y habilidades de los usuarios en las distintas categorías deben ser especificados. Otro aspecto es la descripción de las tareas típicas a realizar por estos usuarios.

- **Requisitos del proceso**

- Definición de funciones dedicadas a la facilidad de aprendizaje o la facilidad de uso, la realización de un modelo a escala o prototipo puede ser considerado como una herramienta que permite el diálogo con los futuros usuarios, aquí, el proceso para establecer y la realización de este diálogo se debe considerar con especial cuidado. - La aplicación de las normas de interfaz de usuario puede ser requerida para obtener un producto de software que se armonizará con otros productos. - la participación de los futuros usuarios en el proceso de validación del software puede ser útil.

Figura 48.Los principales requisitos que se puedan expresar para satisfacer los objetivos de usabilidad

- **Los requisitos de seguimiento del proyecto**

  Los requisitos de seguimiento son bastante limitados.

  Un método consiste en tener especial cuidado con el seguimiento de la especificación de las funciones relacionadas con la usabilidad.

  Otro enfoque puede seguir de cerca el progreso de la validación del prototipo, por ejemplo, el número de comentarios de los usuarios se debe tener en cuenta.

- **Los requisitos de aceptación.**

  Una cuestión clave para la aceptación del producto en cuanto a facilidad de uso de los objetivos es la eficacia de la documentación del usuario. Los requisitos deben tener, en particular, la accesibilidad de la información, por ejemplo, cuadro de contenido, índice, etc. La validación de las funciones de usabilidad es muy importante. Finalmente, la organización de los experimentos con los usuarios puede ser muy eficaz.

Seguridad

Especificar los factores que protegen el software del acceso accidental o malévolo, uso, modificación, destrucción o revelación.

La integridad de los datos recogidos debe ser protegida.

Al final de la fase de especificación de software, el proveedor deberá proporcionar una especificación de seguridad mediante una justificación de informe, cuyo contenido satisfaga los siguientes requisitos:

- Un resumen de los objetivos del sistema de seguridad tal como se describe en la especificación de requisitos del adquirente.
- La descripción y justificación de la elección de las funciones de aplicación de la seguridad.
- La justificación de la idoneidad de este conjunto de funciones, es decir la forma en que son capaces de actuar como contramedidas a las amenazas identificadas.

Al final de la fase de diseño, el proveedor deberá proporcionar un diseño de seguridad mediante la justificación de un informe, cuyo contenido cumplan los siguientes requisitos:

- la descripción y justificación de las decisiones de diseño relacionadas con los componentes que implementan las funciones de aplicación de la seguridad
- La descripción de la separación entre las funciones de aplicación de seguridad y las funciones de seguridad pertinentes.
- La descripción de la aptitud (capacidad de actuar como medida para contrarrestar las amenazas identificadas) y la unión (capacidad de trabajar juntos de una manera que se apoyen mutuamente para proporcionar un conjunto integrado y eficaz) de análisis para el conjunto de componentes de software de la aplicación en la implementación de seguridad de las funciones.
- La descripción de la trazabilidad entre los objetivos de seguridad, las amenazas y los componentes de software.

Mantenimiento

La norma ISO / IEC 9126: 2004 define el "mantenimiento", como "un conjunto de atributos que influyen en el esfuerzo necesario para realizar las modificaciones especificadas."

Se propone, además, las definiciones de sub-características: análisis, variabilidad, estabilidad y la capacidad de prueba.
Para una entidad adquiriente, el mantenimiento es especialmente importante cuando un objetivo es adquirir el producto completo, incluyendo trazas del proceso de desarrollo de software, con el fin de ser capaz de mantener el software después de su entrega.

- **Requisitos del proceso**

El mantenimiento de un producto de software se construye a lo largo del proceso de desarrollo.

- Requiere una fuerte relación entre la trazabilidad de todas las fases del proceso de desarrollo, esto es especialmente importante cuando la evolución del producto se prevé, en esta situación, el responsable tiene que trazar el código que implementa una función que ha de evolucionar.
- Imponer normas de desarrollo para facilitar la legibilidad de los documentos de desarrollo, incluyendo el código fuente, aquí, el enfoque más eficaz es exigir a la organización que se proponga desarrollar su propio conjunto de reglas de programación y para verificar que son satisfactorios: por supuesto, las normas de programación pueden incluir umbrales para las métricas de código fuente.
- Exigir el uso de herramientas tales como: un generador de código y administradores de configuración, sin embargo, la disponibilidad de estas herramientas para la entidad adquiriente, y el hecho de que son utilizados por los profesionales ya es importante.

Figura 49.Los principales requisitos que se puedan expresar para satisfacer los objetivos de mantenimiento

- **Los requisitos de seguimiento del proyecto**

Para efectos de seguimiento, se pueden exigir los siguientes indicadores:

- Para cada fase, el número de componentes puesto en práctica es trazado.
- El número de incumplimientos de las normas de desarrollo para cada fase.
- El número de casos de prueba de regresión producida.

- **Los requisitos de aceptación**

Para los objetivos de mantenimiento, los criterios de aceptación siguientes pueden ser propuestos:

- El grado de trazabilidad entre los documentos resultantes de las fases de desarrollo.
- La satisfacción de las normas de desarrollo.
- La cobertura de los casos de prueba "no regresión.

Portabilidad

Especificar atributos de software que relacionan la facilidad de portar el software a otra computadora (servidor) y/o sistemas operativos.

Definición. La norma ISO / IEC 9126:2004 define "portabilidad" como: **"**Un conjunto de atributos que influyen en la capacidad del software para ser transferido de un entorno a otro."

Las definiciones de sub-características: capacidad de adaptación, facilidad de instalación, la conformidad y reemplazo:

- **Requisitos del proceso**

  La arquitectura de software tiene una importancia muy alta en la satisfacción de los objetivos de la portabilidad. Por ello se recomienda exigir que se tenga especial cuidado en el diseño arquitectónico para considerar los objetivos de la portabilidad.

  Además, las normas de programación se pueden imponer.

- **Los requisitos de seguimiento del proyecto**

  Con el fin de seguir el cumplimiento de los objetivos de la portabilidad durante el desarrollo, la entidad adquirente puede exigir que se suministren informes de revisión relacionados con la fase de diseño con el fin de obtener garantías de que estos objetivos son realmente tomados en cuenta.

  Como un indicador, el número de componentes que tienen una interfaz con el entorno puede ser interesante.

  Por último, el número de incumplimientos de las normas de programación puede ser solicitado.

- **Los requisitos de aceptación**

  Los informes de justificación que resultan de la consideración de los objetivos de la portabilidad durante el diseño deben ser verificados por la entidad adquirente en cuanto a su forma y contenido.

  Por otra parte, la satisfacción de estándares de programación es un tema importante.

4.2.2.4 Organizar los requisitos específicos

El método para organizar los requisitos detallados con frecuencia se relacionan con la arquitectura probable de la aplicación. Por ejemplo, si el diseño debe orientarse a objetos, se organizará por casos de uso o por clase debe considerarse debido a que facilita la trazabilidad. Si la naturaleza de la aplicación lleva una descomposición en funciones, entonces organizar los requisitos por jerarquía funcional.

El propósito de la SRS se logra mediante la organización de los requisitos del sistema en categorías conceptuales. En la práctica, es difícil de identificar y separar las necesidades de otros aspectos de la percepción del cliente del sistema que a menudo se incluyen en los documentos que definen los "requisitos. El analista debe capturar el estado de las necesidades básicas del cliente y la comunidad técnica, propiamente requisitos de forma, y organizar o agruparlos en categorías significativas.

Aunque la organización de las declaraciones de los usuarios no estructurados en un conjunto estructurado de las necesidades, el analista debe identificar los requisitos técnicos sin ser desviados a declarar un método de implementación. Que se distraiga en cuestiones de aplicación antes de una comprensión clara de los requisitos puede conducir tanto a una insuficiencia de requisitos y a una implementación defectuosa. Discernir entre los requisitos técnicos y técnicas puestas en práctica es un desafío constante para el analista.

Modo del sistema

Es el modo de operación. Algunos sistemas se comportan de un modo muy distinto en función del modo en el que estén.

La descripción del sistema debe ser expresada en términos operacionales y logísticos. Los temas tratados incluyen en el sistema: capacidades operativas, características físicas, parámetros de rendimiento y los valores esperados, las interfaces y las interacciones con su entorno, los requisitos de documentación, los requisitos de fiabilidad, las necesidades logísticas y las necesidades de personal.

- Identificar los requisitos que se derivan de otros requisitos.
- Organizar las necesidades en diferentes niveles de detalle en sus niveles apropiados.
- Comprobar la integridad de los requisitos.
- Identificar inconsistencias entre los requisitos.
- Identificar claramente las capacidades, condiciones y limitaciones para cada necesidad.
- Desarrollar un entendimiento común con el cliente y los objetivos de los requisitos.
- Identificar los requisitos que se completa en la SRS.

Tabla 146.Pautas para mantener los requisitos de una manera estructurada tanto para los clientes como el equipo técnico.

Es importante que la estructura se añade al conjunto de las necesidades de los analistas, y que las representaciones de la especificación de los requisitos del software se comuniquen de una manera estructurada.

Clases de usuario

En función de a quién esté dirigido interesará resaltar unas funciones u otras.
Es un estilo orientado a objetos. En esta organización los requisitos se agrupan en clases. Si el diseño debe orientarse a objetos, la organización por casos de uso o por clases debe considerarse debido a que facilita la trazabilidad.

- **Organización de los requisitos específicos del software por casos de uso:**

  El proceso unificado de desarrollo de software (USDP) explota la observación de que muchos requisitos ocurren de manera natural en secuencias operativas. Los casos de uso, algunas veces llamados escenarios (en UML, un escenario es una instancia de un caso de uso).
  El proceso unificado de desarrollo de software favorece la organización de los requisitos por casos de uso.

  Organizarlos de esta manera es útil para producir casos de uso más grande a partir de los pequeños. Un caso de uso especifica una secuencia de acciones, incluyendo variantes que el sistema puede realizar y que ofrece un resultado observable o tangible para un determinado usuario.

  Los casos de uso recogen los requisitos funcionales (qué queremos que haga) y los requisitos no funcionales (restricciones de tiempo, sistema operativo, etc.) del sistema.

1. Actor: Un actor es un conjunto coherente de roles que desempeñan los usuarios de los casos de uso cuando interactúan con estos. Un actor representa a un tipo de usuario cuando este interacciona con el sistema en un caso de uso. Existen otro tipo de actores que son todos los sistemas externos que en algún momento interaccionan con el sistema.
2. Caso de uso: Cada una de las maneras en que un usuario usa el sistema está representada por un caso de uso. Lo importante es que cada caso de uso ofrece un valor concreto, un resultado de utilidad para sus actores.
3. Modelado de casos de uso: Es un modelo del sistema que comprendía los casos de uso, sus actores y sus relaciones. Constituye la base del acuerdo entre el cliente y los desarrolladores del sistema sobre qué debe realizar el sistema y cómo debe hacerlo, sirve como entrada para los flujos de análisis, diseño y pruebas

Figura 50.Captura de requisitos vía casos de uso

- **Organización de los requisitos específicos del software por clases:**

  La atención se centra en un estilo orientado a objetos para organizar los requisitos, donde primero se identifica una clasificación, equivalente a seleccionar las clases y después se colocan los requisitos individuales en las categorías o clases obtenidas. Existen dos enfoques para realizarlo:

  - Desde un punto de vista, mira las clases como una herramienta para organizar los requisitos, pero no necesariamente se considera que se puede utilizar para el diseño real.
  - Usa las clases desarrolladas para los requisitos en el diseño y la implementación orientados a objetos reales. Este enfoque fomenta una trazabilidad uno a uno de los requisitos específicos a los métodos donde cada requisito específico funcional debe corresponder a una función en leguaje que se usa.

Ventajas	Desventajas
- Organizar los requisitos por clases que se utilizarán en el diseño es que fomenta una correspondencia estricta entre los requisitos, el diseño y la implementación. - Probabilidad de que se reutilicen las clases que corresponden a los conceptos del mundo real que las que no lo hacen.	- Riesgo de cambiar más adelante las clases usadas, con lo que se destruye la correspondencia entre los requisitos y la selección de clases. - Se requiere que se seleccionen clases cerca del comienzo del ciclo de desarrollo, y muchos argumentos que, en efecto, el diseño se realiza al hacerlo.

Tabla 147.Ventajas y desventajas del método de clasificación de cada requisito específico

Identificación de clases

Se identificarán las clases de domino de la aplicación. El uso de clases de dominio es una manera de organizar, reflexionar y rastrear los requisitos. La meta es identificar un conjunto mínimo pero suficiente de clases de dominio que incluya todos los requisitos específicos. Los casos de uso son una fuente primordial de clase de dominio. A menos que la aplicación sea en esencia una interfaz gráfica de usuario, entonces es indispensable posponer la introducción de clases de interfaz gráfica de usuario hasta la etapa de diseño, siempre que sea posible. Esto se debe a que éstas no son clases de dominio y los diseñadores tienden a cambiar con frecuencia.

- Desarrollar una colección amplia, sin solapar los casos de uso.
- Crear un diagrama de secuencia para cada caso de uso.
- Reunir las clases utilizadas en los diagramas de secuencia.
- Determinar las clases de dominio esenciales.
- Clasificar los requisitos funcionales detallados según estas clases.
    - Enumerar cada atributo como un requisito.
    - Enumerar cada objeto específico de la clase.
    - Enumerar cada función requerida de los objetos en la clasificación.

Figura 51.Seleccionar clases de dominio para los requisitos que determinan la clasificación

Objetos

Se pueden agrupar las funciones en términos de las suministradas por jerarquías de objetos.

Las aplicaciones requieren la existencia de entidades (instancias u objetos).

- Es establecerlos en la clase que crea el objeto.
    - Ventaja de este enfoque: Hace coincidir parte del código.
    - Desventaja de este enfoque: Cambia la decisión acerca de que objeto crea el objeto requerido y por ello los requisitos tendrían que desplazarse. Aunque otros objetos no crean el objeto requerido, puede hacer referencia a él más a menudo que el objeto creador.
- Introducir una clase especial que agrega los objetos requeridos de la clase relevante. Esto puede resultar complicado, además de que supondría la adición de clases innecesarias.
- Enumerar los requisitos de objetos con las clases a las que pertenecen

Figura 52.Opciones donde deben establecerse los requisitos.

Rasgo

Son servicios observables externamente que puede requerir una secuencia de entradas y salidas en un orden concreto.

Una secuencia usual para obtener los requisitos específicos funcionales se centra al estilo orientado a objetos.

- El proceso comienza por enumerar las clases mencionadas en los casos de uso.
- La colección resultante de clases casi siempre está incompleta y debe hacerse un esfuerzo para descubrir las clases de dominio restantes. Después se inspecciona la colección de clases.
- Para cada una de las clases obtenidas, el analista escribe toda la funcionalidad que requiere la aplicación, sobre todo la que pertenece a la clase. Esto se logra en la forma de atributos y funciones.
- Cada objeto de la clase requerido que se conoce debe estar enumerado como un requisito de esa clase.
    - Los eventos que deben manejar los objetos de la clase deben estar especificados
    - Los requisitos específicos se inspeccionan conforme avanza el proceso.
    - Los planes de pruebas para cada requisito específico deben crearse al mismo tiempo.
- Se inspeccionan los requisitos específicos contra los requisitos del cliente.
- Los requisitos específicos se verifican con el cliente y después se liberan.

Figura 53.Secuencia para obtener la SRS funcionales

Un error común en la clasificación de requisitos por clase es tratar el proceso como si fuera diseño. El lenguaje utilizado debe ser en español.

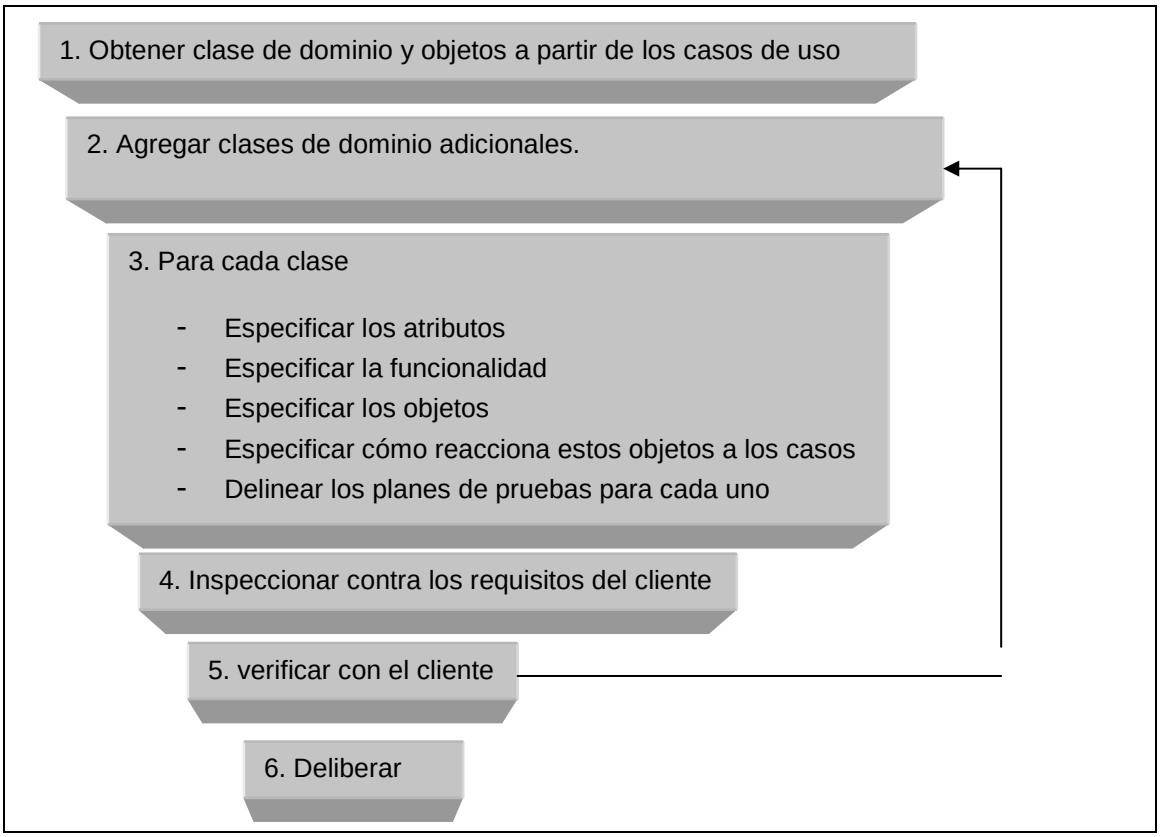


Figura 54.Resumen de los pasos de la figura 53

Estímulo

Algunos sistemas se describen mejor en términos de estímulo - respuesta.

Los requisitos comienzan como ideas o conceptos que puedan derivarse de una respuesta a una amenaza percibida de compartir la seguridad o el mercado, de una imposición de una legislación o regulación, desde un deseo de crear un sistema nuevo, desde la necesidad de reemplazar un sistema existente, o de alguna otra necesidad real o percibida.

Aunque las ideas o conceptos pueden originarse en un individuo, un conjunto de requisitos por lo general se obtiene mejor de la interacción de un grupo donde se comparten las ideas y la forma.

- Talleres estructurados.
- Sesiones de reflexión o de resolución de problemas.
- Las entrevistas.
- Las encuestas/cuestionarios.
- Observación de los patrones de trabajo (por ejemplo, estudios de tiempo y movimiento).
- La observación del clima organizacional y política del sistema.
- La revisión de documentación técnica.
- Análisis de mercado.
- Evaluación del sistema competitivo.
- La ingeniería inversa.
- Las simulaciones.
- Prototipos
- Procesos de evaluación comparativa y los sistemas.

Figura 55.Técnicas para la identificación de requisitos en términos de estímulo – respuesta.

Contestación

Clasificación en función del tipo de respuesta que da el sistema.

Jerarquía Funcional

Se trata de organizar el sistema en términos de funciones con entradas y salidas similares.

Antes de entrar a explicar con detalle la jerarquía funcional, vamos a explicar el concepto de modelo. Un modelo conceptual es aquello que nos permite conseguir una abstracción lógica-matemática del mundo real y facilita la comprensión del problema a resolver.

Los objetivos que deben cubrir los modelos se pueden concretar en los siguientes:

- Facilitar la comprensión del problema a resolver.
- Establecer un marco para la discusión, que simplifique y sistematice la labor tanto del análisis inicial como de las futuras revisiones del mismo.
- Fijar las bases para realizar el diseño.
- Facilitar la verificación del cumplimiento de los objetivos del sistema.

Modelo jerarquizado

Cuando un problema es complejo, la primera técnica que se debe emplear es descomponerlo en otros más sencillos de acuerdo con el viejo axioma “divide y vencerás”. De esta manera se establece un modelo jerarquizado en el que el problema global queda subdividido en un cierto número de subproblemas. Este tipo de descomposición se denomina horizontal y se trata de descomponer funcionalmente el problema.

A continuación, en el análisis de cada uno de estos subsistemas se pueden emplear las mismas técnicas que con el sistema global. Por lo tanto, es posible volver a descomponer cada uno de estos subsistemas a su vez en otros más simples. Para completar el modelo se tendrán que establecer las interfaces entre las partes o subsistemas en que queda dividido, para posibilitar el funcionamiento del sistema global.

Cuando se descompone un modelo tratando de detallar su estructura, este tipo de descomposición se denomina vertical. Esta técnica supone aplicar el método de refinamiento sucesivo al modelo del sistema.

Hay varios esquemas para la organización de las especificaciones en un conjunto ordenado. El esquema más utilizado es el de reunir los requisitos en una jerarquía de capacidades. En las capacidades más generales se descomponen los requisitos de subordinación, muestra la relación entre todos los requisitos de más bajo nivel. En el esquema de un modelo de jerarquía en la SRS debe indicar las relaciones entre los requisitos. Las relaciones específicas que dependerán de los métodos, técnicas y herramientas utilizadas para capturar, registrar y almacenar los requisitos.

Algunas de las relaciones entre los requisitos que se pueden mantener en una SRS incluyen las dependencias jerárquicas siguientes:

- Eventos.
- La información / los datos.
- Los objetos físicos o abstractos.
- Funciones.

4.2.2.5 Conclusiones generales

- En el desarrollo de un proyecto, las primeras fases han sido descritas en términos de establecer las necesidades del cliente, evaluar la viabilidad del sistema y en la creación del documento de especificación de requisitos.

- El plan de administración de proyectos de software es el medio más importante para guiar la administración de proyectos. Un aspecto clave es la estimación de los costos del proyecto, proceso que requiere una revisión continua durante toda la vida del proyecto.

- Del mismo modo que los usuarios son cada vez más consistentes en sus derechos, todo participante en un proceso de suministro o fabricación debe conocer sus deberes, para asegurar y minimizar sus responsabilidades y riesgos.

- Un dominio actualizado de las responsabilidades legales básicas resulta imprescindible hoy en día. Aunque cada país parte de una normativa legal específica, el comercio internacional tiende a unificar las leyes en amplias zonas geográficas.

- El enfoque propuesto en este capítulo de requisitos permite al adquirente obtener los requisitos de calidad de los objetivos de calidad como resultado de un análisis de riesgo sobre la base de las características del producto de software.

- El objetivo no es tratar de obtener mediciones directamente. Se siguiere que los requisitos se relacionen con los métodos de desarrollo de software, en la elaboración de informes de justificación, sobre el seguimiento del desarrollo del proyecto y en la aceptación del producto.

4.3 Descripciones de diseño del software IEEE Std 1016 – 2009

4.3.1 Información general

4.3.1.1 Ámbito de aplicación

Esta norma describe los diseños de software y establece el contenido de la información y la organización de una descripción del diseño de software (SDD). El SDD es una representación de un diseño de software que se utilizará para el registro de información sobre el diseño y la comunicación de la información de diseño a las partes interesadas claves en el diseño.

4.3.1.2 Propósito

Esta norma específica los requisitos sobre el contenido de la información y la organización de las SDD. La norma específica los requisitos para la selección de lenguajes de diseño que se utilizarán para SDD, y los requisitos para la documentación de puntos de vista de diseño que se utilizarán en la organización de un SDD.

4.3.1.3 A quién va dirigido

Esta norma está destinada a los interesados técnicos y de gestión de preparar y usar las SDD. Esto lleva a un diseñador en la selección, organización y presentación de información sobre el diseño. Para una organización de desarrollo de sus propias prácticas SDD, el uso de esta norma puede ayudar a asegurar que las descripciones de diseño sean completas, concisas, coherentes, intercambiables, apropiadas para el registro de experiencias y fácil de comunicar.

4.3.2 Definiciones.

Todas las definiciones que se exponen a continuación esta explicadas con detalle a lo largo de las secciones siguientes.

Atributo de diseño: Un elemento de una vista de diseño, nombres con característica o propiedad de una entidad de diseño, la relación de diseño, o la restricción de diseño.

Consideraciones de diseño: Un área de interés con respecto a un diseño de software.

Restricción de diseño Elemento de una vista de diseño con nombres y especifica una regla o restricción a una entidad de diseño, atributos de diseño, o una relación de diseño.

Elemento de diseño Un elemento que ocurre en una vista de diseño que puede ser cualquiera de los siguientes: diseño de la entidad, la relación de diseño, atributos de diseño, o la restricción de diseño.

Entidad de diseño: Un elemento de una vista de diseño es estructural, funcional o de otra manera distinta de otros elementos, o juega un papel diferente en relación con las entidades de diseño.

Superposición de diseño: Una representación de la información de diseño adicional y detallada, con miras a una vista de diseño existentes.

Justificación del diseño: La captura de información, el razonamiento del diseñador que llevó al sistema tal como fue diseñado, incluidas las opciones de diseño, compensaciones, las decisiones tomadas y las justificaciones de las decisiones.

Relación de diseño: Elementos de una vista de diseño, los nombres de una conexión o correspondencia entre las entidades de diseño.

Partes interesadas del diseño: Un individuo, organización o grupo que tiene un interés o preocupación en relación con el diseño, el diseño de algún elemento de software.

Tema de diseño: Un elemento de software o sistema por el que un SDD estará preparado.

Diseñador: El participante responsable de elaborar y documentar el diseño de software.

Vista de diseño: Una representación compuesta por uno o más elementos de diseño para hacer frente a un conjunto de problemas de diseño desde un punto de vista de diseño especificado.

Punto de vista del diseño: La especificación de los elementos y convenciones para la construcción y el uso de una vista de diseño.

Diagrama (tipo) **o esquema**: Un fragmento lógicamente coherente de una vista de diseño, con iconos gráficos seleccionados y las convenciones para la representación visual de un lenguaje de diseño asociadas, que se utilizará para representar los elementos seleccionados de diseño de interés para un sistema en el diseño de un único punto de vista.

4.3.3 Modelo conceptual para la descripción del diseño de software

Esta cláusula establece un modelo conceptual SDD. El modelo conceptual incluye términos y conceptos básicos de la SDD el contexto en el que SDD se preparan y utilizan, los actores que las utilizan, y cómo se utilizan.

4.3.3.1 Diseño de software en contexto o marco

Un diseño es una conceptualización de un tema de diseño (en fase de diseño del sistema o software en fase de diseño), demuestra un medio para satisfacer sus necesidades; sirve de base para el análisis y la evaluación y se puede utilizar para guiar su aplicación.

Desarrollo del proceso	Razones de adopción	Dominios típicos
Lineal (por ejemplo, "cascada")	Necesidad de determinar el tiempo de entrega si es aceptable.	Muchos de los sistemas se desarrollan de esta manera. Este enfoque se ha utilizado muy ampliamente y todavía es apropiado para sistemas que necesitan demostrar cualquier tipo de alta integridad.
Incrementales	Puede haber una necesidad de demostrar la viabilidad de una idea, establecer una posición en el mercado rápidamente, o averiguar si un mercado puede existir.	Ampliamente utilizado para paquetes, juegos, sistemas operativos. Además, hay una necesidad de una estrecha interacción con los usuarios finales durante el desarrollo.
Reactivo	Donde la evolución de un sistema es en gran parte la respuesta a lo que los desarrolladores distinguen como necesario.	El software de código abierto de todo tipo generalmente se desarrolla de esta manera, al igual que muchos sitios web.

Tabla 148.Algunos ejemplos de los procesos de desarrollo del software

Las dos principales formas: modelo de cascada y el enfoque de prototipos. Al igual que con los procesos, lo mismo sucede con las tareas. Mientras que la terminología utilizada en las diferentes fuentes puede variar un poco, a continuación se describen un conjunto de prácticas:

- Estudio de Factibilidad: El papel de este estudio es para explorar si una solución es posible en el conjunto de restricciones. Así como un estudio da algún pensamiento previo a los problemas de diseño, aunque sólo sea a nivel de arquitectura (qué tipo de solución podría ser más práctico).

- Captura de requisitos: El objetivo es identificar lo que los usuarios finales y las necesidades del sistema previsto están obligados a cumplir.

A veces se sugiere un enfoque ideal para esta tarea para que sea completamente independiente de cualquier consideración eventual de diseño e implementación. Sin embargo, esto es cada vez menos apropiado, especialmente cuando un sistema necesita interactuar con otros, y en la reutilización de software existente.

- Análisis: Esta etapa suele ser vista como un modelo más formal del problema, es decir, las necesidades identificadas en los documentos producidos a partir de la etapa de captura de requisitos. Esto constituye un aporte importante al proceso de diseño, y puede derivar en decisiones basadas en soluciones "supuestas".

- Diseño Arquitectónico: Preocupación por la forma general de la solución que se adopte. Por ejemplo, si se utiliza un sistema distribuido, los elementos más importantes "son" y "cómo" van a interactuar.

- Diseño detallado: El desarrollo de las descripciones de los elementos identificados en la anterior fase, involucran la interacción con este. También hay bucles de retroalimentación en las fases anteriores, los diseñadores deben asegurarse que el comportamiento de su solución cumpla con los requisitos.

- Codificación (implementación): Esto puede implicar la elaboración de datos, estructuras de comunicación y escritura de grandes cantidades de código para unir los componentes existentes en conjunto.

- Unidad y pruebas de integración. Esta fase implica la validación del sistema implementado en contra de los requisitos originales, producidos a partir de la especificación de la fase de análisis y el diseño en sí.

- Operación y mantenimiento: La operación algunas veces puede parecerse a una fase de pruebas extendida, que proporcione información a los usuarios de muchas maneras.

  El mantenimiento es una cuestión bastante y diferente puede implicar la repetición de todas las tareas de las fases anteriores y, de hecho, la integridad del sistema. En otras palabras, la coherencia en el plan de diseño o bien podría ser una limitación importante en la tarea de mantenimiento.

Evaluar el proceso de diseño

Todo producto de diseño surge como resultado de algún tipo de proceso de diseño. La calidad del proceso de diseño es un factor importante que debe tenerse en cuenta cuando se trata de entender los problemas de diseño de calidad.

Las tareas que participan en el desarrollo del software, tales como la codificación y prueba, las medidas actuales de productividad y de calidad son bastante limitadas. Se realiza un estudio de los problemas de calidad aplicada a los procesos de desarrollo de software.

En particular, dentro de su modelo de madurez que se producen en los procesos que una organización utiliza para la producción de sistemas se encuentran:

- Inicial: Pocos procesos están definidos, y el éxito de un proyecto depende del esfuerzo individual.

- Repetible: Los procesos básicos de gestión de proyectos son establecidos y utilizados para realizar seguimiento de costes, calendario y funcionalidad.

- Definidas: El proceso del software está documentado, estandarizado e integrado en un proceso de software en toda la organización. Todos los proyectos utilizan una versión documentada y aprobada del proceso de la organización para el desarrollo y mantenimiento de software.

- Administrados: Tanto el software de proceso y de productos son cuantitativamente entendidos y controlados con medidas detalladas.

- La optimización: Mejora continúa del proceso, está habilitado por la retroalimentación cuantitativa del proceso y de las pruebas de ideas innovadoras y tecnológicas.

Para mejorar la calidad del proceso de diseño es necesario encontrar la forma de incluir de entrada en el diseño actividades que pueden proporcionar:

- Conocimiento del dominio sobre el tipo de problema planteado, y aspectos importantes de cualquiera de las funciones de ejecución.

- Métodos de conocimientos que ayuden a comprender las técnicas de diseño que se utiliza.

Experiencia de proyectos similares

Hay tres formas ampliamente adoptadas para proveer estos insumos en el proceso de diseño.

1. Las revisiones técnicas: Su utilización permite al equipo de diseño obtener conocimiento de las experiencias de los demás en particular de cualquier conocimiento del dominio y métodos que ellas podrían tener. Esta forma de entrada está destinada en gran parte a mejorar el diseño del producto o puede tener un impacto significante en el proceso de diseño mediante la identificación de técnicas útiles o anotaciones que pueden ser aplicables a un problema particular.

2. Reseñas o revisiones de gestión: Se refiere principalmente a desarrollar y refinar estimaciones de esfuerzo y plazos para el proyecto en su conjunto, y con la recopilación de los datos pueden ser necesarios para tales estimaciones. La tarea de estimación se basa en gran medida en el conocimiento del dominio y de la experiencia, ya que se requiere el uso de estos para hacer las proyecciones necesarias.

3. Los prototipos: Proporcionan los medios para obtener el conocimiento del dominio y algunas formas de experiencia acerca de un problema en particular.

De estos tres, la revisión de la gestión implica la realización de algún tipo de evaluación de la forma en que un diseño se está desarrollando, y puedan, proporcionar un marco para el uso de las otras dos formas.

El uso de la información y mecanismos de seguimiento tendrá un papel importante en el control de problemas de calidad y la garantía de que estas cuestiones son en realidad tratadas durante la revisión. La calidad de la documentación juega un papel importante en el establecimiento de un proceso de alta calidad mediante la reutilización de la experiencia.

4.3.3.2 Descripciones de diseño dentro del ciclo de vida

En esta norma, se utiliza para describir el diseño de diversas situaciones en las que un SDD se puede crear y utilizar.

4.3.3.2.1 Influencias en la preparación de SDD

La clave del ciclo de vida del producto de software que conduce un diseño de software suele ser la especificación de requisitos software (SRS). Un SRS captura los requisitos de software que impulsarán las limitaciones de diseño

La especificación de los requisitos del software es una etapa preliminar donde los requisitos del sistema incluyen a la empresa, organización, usuarios, es decir, los interesados. La necesidad de los interesados en definir un sistema proporciona servicios requeridos por los usuarios incluyendo deseos, necesidad, expectativas, anhelos, limitaciones, decisiones técnicas.

La especificación de requisitos incluye los siguientes aspectos:

- Especificaciones técnicas.
- Especificaciones de usabilidad.
- Funciones sistema-nivel.
- Requisitos de seguridad.
- Criticidad máxima y prioridad de los requisitos.
- Limitaciones de rendimiento mínimo.

- Referencias relacionadas al diseño del sistema pruebas y normas.
- Descripción de los métodos y herramientas utilizadas para definir la trazabilidad de los requisitos de la arquitectura del sistema.
- Supuestos de productos y dependencias.
- Funciones del producto y los requisitos funcionales del sistema.
- Requisitos del sistema de calidad, características de calidad y seguridad
- Requisitos de factores humanos (ergonomía).
- Requisitos para las interfaces internas y externas con el sistema, hardware, software, comunicaciones, usuarios etc.
- Requisitos del diseño del sistema y restricciones de diseño.
- Sistema de pruebas y requisitos de cualificación.
- Requisitos de aceptación.
- Requisitos de adaptación al sitio.
- Requisitos para la documentación de usuario y formación.
- Requisitos de empaque, instalación, operación, productos de actualización y mantenimiento.

Se pueden incluir en el sistema requisitos de infraestructura propicia para la organización, incluidos los recursos y herramientas.

La especificación de requisitos se basa en la definición del sistema. Los requisitos de alto nivel se especifican durante la planeación inicial, se detallan y se hacen específicos para poder definir las características del producto de programación que deben contener.

La especificación de requisitos establecerá que es el producto sin implicar como es este, es decir, el diseño del producto se refiere a cómo se van a proporcionar las características deseadas.

- En las interfaces externas las características externas del producto de programación incluyen despliegues que verá el usuario, formato de reportes, opciones de reportes, diagrama de flujos de datos, diccionario de datos etc.

- Los requisitos funcionales se expresan en notaciones relacionales y están orientadas a estados que especifican las relaciones entre entradas, acciones y salidas.

- Los requisitos de operación contemplan las siguientes características tales como el tiempo de respuesta de varias actividades, tiempo de procesamiento de varios procesos, número de procesos ejecutados, restricciones de memoria principal y secundaria, elementos particulares como restricciones de seguridad, características de confiabilidad no comunes.

- El manejo de excepciones: Incluyen acciones a tomar y los mensajes a enviar en respuestas a situaciones no deseadas; se debe tener una tabla de excepciones y las respuestas donde se incluyan todas las categorías posibles sin excluir el fallo temporal o permanente de algún recurso, datos de entrada, valores internos, parámetros incorrectos,

inconsistente fuera de rango, violación de los límites de capacidad, violaciones en las operaciones.

- Los subconjuntos iniciales y prioridades de instrumentación: Es importante especificar las prioridades de instrumentación de las distintas capacidades del sistema; las características esenciales deseadas y sugeridas se deben identificar para proporcionar una guía a los diseñadores e instrumentadores.

- Los productos de programación desarrollan una serie de versiones sucesivas, cada versión sucesiva puede incorporar las capacidades de versiones previas, así como nuevas funciones. El cliente puede pedir entregas parciales de subproductos y versiones sucesivas que pueden mejorarlas.

- Modificaciones y mejoras previsibles: Las modificaciones previstas para el producto pueden ocurrir como resultados de cambios inesperados en el presupuesto o en el objetivo del producto. Es importante conocer e incorporar en la especificación cada uno de los cambios.

- Criterios de aceptación del producto: Especifican las pruebas funcionales y de rendimiento que se deben realizar y los estándares que se aplicarán en el código fuente, documentación interna y documentos externos tales como especificación del diseño, plan de pruebas, manual de usuario, operaciones, procedimiento de instalación y mantenimiento.

- Guías y sugerencias de diseño: La especificación de requisitos se refiere a aspectos funcionales y de rendimiento del software y hace hincapié en la especificación de las características del producto, sin determinar cómo se lograrán estas características. El cómo de la instrumentación del producto es tema del diseño. Estas deben ser registradas como sugerencias y guías para los diseñadores del producto y no como requisito para el diseño del producto.

- Glosario de términos: Proporciona definiciones de términos que pueden ser extraños para el cliente y los diseñadores. En particular se deben definir términos estándar utilizados de manera no estandarizada.

4.3.3.2.2 Influencias en productos software del ciclo de vida

El SDD influye en el contenido de varios productos importantes del ciclo vida de software de trabajo. Se contemplan: requisitos de la especificación de software y prueba documentación.

Además de lo expuesto en el apartado anterior 4.3.3.2.1 influencias en la preparación de la descripción del diseño del software explicadas con detalle en la especificación de requisitos del software. A continuación detallaremos el documento de diseño de pruebas.

El documento del diseño de pruebas es indispensable para una buena organización, asegurar la reutilización ya que a través de ella se puede optimizar tanto la eficiencia como la eficacia.

Podemos contemplar distintas fases de pruebas:

- La planificación, cada estrategia de prueba para el producto.
- Especificación del diseño de prueba surge de la extensión y en detalle el plan de pruebas.
- En el diseño de prueba definir de una manera detallada cada uno de los casos.
- Generar casos de prueba detallados, es decir, procedimientos de prueba.
- Especificaciones de casos de prueba, especificaciones de los procedimientos. Los procedimientos determinan como se desenvuelve la ejecución.

- Plan de pruebas

  Se contempla la siguiente estructura fijada en el plan de pruebas.

  - Identificador del documento.
  - Resúmenes de elementos y características a probar.
  - Elementos de software a probar.
  - Características y no características que se van a probar y no probar.
  - Enfoque general de la prueba.
  - Criterios de fallo para cada uno de los elementos.
  - Documentos a entregar.
  - Necesidades de entorno y personal y formación.
  - Preparación, actividades y ejecución de pruebas.
  - Gráficos de tiempo.
  - Riesgos y planes de contingencia para cada uno de los riesgos.
  - Aprobaciones, firma, nombre y puesto que desempeña.

- Especificación del diseño de pruebas

  Se contempla la estructura fijada en él estándar especificación del diseño de pruebas:

  - Identificador para la especificación.
  - Características de los elementos software que se van a probar.
  - Detalle sobre el plan de prueba en el que emerge el diseño incluyendo técnicas de prueba y métodos de análisis de los resultados.
  - El identificador de cada prueba debe contemplar: identificador, casos que se van a usar, procedimientos que se van a tomar.
  - Criterios de fallo de pruebas, determina si una característica o combinación ha pasado con éxito la prueba pertinente.

- Especificación de caso de prueba

  Se contempla la estructura fijada en el estándar, especificación de caso de prueba.

  - Identificador de la especificación.
  - Elementos del software que se van a probar y características que se ejecutaran en cada caso.
  - Especificaciones de las entradas para ejecutarlas en cada caso.
  - Especificaciones de salida y características específicas para los elementos que se van a probar.
  - Necesidades de entorno.
  - Requisitos de procedimientos.
  - Dependencias entre los distintos casos.

- Especificaciones de procedimiento de prueba

  Se contempla la estructura fijada en el estándar especificación de procedimiento de prueba.

  - Identificador único.
  - Objetivo del procedimiento y los casos que se van a ejecutar.
  - Requisitos para la ejecución.
  - Pasos en el procedimiento.
  - Secuencia necesaria de acciones.
  - Acciones de inicio para la ejecución.
  - Acciones durante la ejecución.
  - Puntos para el reinicio de la ejecución.
  - Acciones para detener la ejecución.
  - Acciones para tratar los acontecimientos extraños.

4.3.3.2.3 Verificación del diseño y el papel del diseño en la validación

La verificación es la determinación de si un producto de trabajo de software cumple unos requisitos específicos.

La verificación es un concepto fundamental en el diseño de la programación ya que el diseño es el puente entre los requisitos del cliente y la instrumentación que satisface esos requisitos.

Un diseño es verificable si podemos demostrar que el diseño generará el producto que satisface los requisitos del cliente, esto se desarrolla en los siguientes pasos:

- Verificación, la definición de los requisitos de programación satisface las necesidades del usuario.
- Verificación de que el diseño satisface la definición de los requisitos.

Es necesario subrayar que los planes de prueba son un producto del proceso del diseño. El plan de prueba debe estar acoplado a las especificaciones demostrables del sistema. En la teoría de control se habla de los estados de un sistema como observables y controlables; en el diseño de la programación se debe de estar seguro de que el sistema está estructurado para que los estados internos puedan ser observados y probados, y los resultados tengan relación con los requisitos.

El propósito de la verificación del software es la comprobación de que cada producto de trabajo del software o servicio refleje los requisitos especificados.

Como resultado del proceso de verificación del software se obtiene lo siguiente:

- Una estrategia de verificación se ha desarrollado al igual que se ha implementado.
- Criterios para la verificación e identificación de trabajo de todos los productos del software.
- Tareas requeridas de verificación para llevar a cabo.
- Las deficiencias son identificadas y registradas.
- Los resultados de las actividades obtenidas se podrán a disposición del cliente así como de las partes interesadas.

Las tareas se pondrán en práctica de acuerdo con políticas y procedimientos con respecto al proceso de verificación del software. En el proceso de implementación se determinará si el proyecto merece un esfuerzo de verificación y grado de independencia.

Los requisitos serán analizados por la criticidad el cual puede ser medido en términos de:

- Error no detectado en el sistema o requisito que puede causar: fracaso del producto, pérdida de equipamiento económica, etc.
- Vencimiento de los riesgos con la tecnología del producto que se usará.
- Disponibilidad de fondos y recursos.

Con base en el alcance, la magnitud, complejidad y criticidad en el análisis de los objetivos del ciclo de vida de las tareas y verificación de productos del software se determinará, incluyendo métodos asociados, técnicas y herramientas para realizar tareas del ciclo de vida útil de los productos del software.

En base a las tareas de verificación se determina un plan de verificación que será desarrollado y documentado. El plan se ocupará de las tareas del ciclo de vida; las tareas requeridas de verificación para cada actividad del ciclo de vida y productos software y recursos relacionados, responsabilidades y el calendario, además se llevará a cabo, y problemas y no conformidades detectadas mediante la verificación la cual se hace constar en el proceso de problemas del software. Todos los problemas y no conformidades deberán ser resueltos.

En los requisitos se verificará teniendo en cuenta los siguientes criterios:

- Los requisitos del sistema son: consistentes, viables y comprobables.
- Los requisitos del sistema de acuerdo con los criterios de diseño que han sido asignados: elementos hardware, elementos software y manual de operaciones.
- Los requisitos software deben estar relacionados con la seguridad y criticidad.

El diseño de verificación deberá ser verificado dependiendo de los siguientes criterios:

- El diseño es correcto, consistente con los requisitos y trazable.
- El diseño implementa la secuencia correcta de eventos, entradas, salidas, interfaces, flujo lógico, asignación de tiempo, tamaño de los presupuestos, definición de error, aislamiento y recuperación.
- El diseño seleccionado puede derivarse de los requisitos.

El código de verificación se verificará dependiendo de los siguientes criterios:

- El código es trazable y cumple con los requisitos y normas de codificación.
- El código implementa la secuencia de eventos correctos, interfaces consistentes, datos correctos, control de flujo e integridad, asignación de tamaño, error, aislamiento y recuperación.
- El código seleccionado puede derivarse del diseño o de los requisitos.

La integración de la verificación se verificará dependiendo de los siguientes criterios:

- Los componentes del software y las unidades de cada elemento software han sido completos e integrados correctamente.
- Los elementos hardware, elementos software y manual de operaciones han sido integrados en el sistema.
- Mediante un plan de integración se realizarán las tareas de integración.

La documentación de la verificación se verificará dependiendo de los siguientes criterios:

- La documentación es adecuada, completa y consistente.
- La preparación de la documentación es oportuna.
- La gestión de la configuración de los documentos sigue procedimientos especificados.

El proceso de validación del software es para comprobar que los requisitos mediante un uso específico se han cumplido.

Los resultados de la implementación en el proceso de validación del software son:

- Una estrategia de validación es desarrollada e implementada.
- Los problemas son identificados y registrados.
- Las tareas se llevan a cabo.
- Demostración de que los productos del software son desarrollados para su uso previsto.
- Los resultados de las tareas de validación se pondrán a disposición del cliente y otras partes interesadas.

Un plan de validación deberá ser desarrollado incluyendo los siguientes aspectos:

- Temas de validación.
- Tareas de validación a realizar.
- Recursos, responsabilidades y calendarios para su validación.

En el plan de validación se llevarán a cabo problemas y no conformidades detectadas donde constará en el proceso de resolución de problemas. Todos los problemas y no conformidades deberán ser resueltos. La validación constará de las siguientes tareas: pruebas, análisis, modelado, simulación, requisitos de prueba, casos de prueba, especificaciones de prueba, análisis de prueba. Las revisiones del software se encuentran en la gestión de proyectos tanto técnicos como los niveles que se llevan a cabo durante la vida del proyecto.

Los resultados de la implementación de la revisión del programa son:

- Revisiones técnicas y de gestión que se llevan a cabo sobre las bases del proyecto.
- El estado y los productos de una actividad se evalúan a través de las tareas de revisión.
- Revisar los resultados y darlos a conocer a todas las partes interesadas.
- Los elementos de acción obtenidos mediante los resultados de las revisiones se realizará un seguimiento hasta el cierre.
- Los riesgos y los problemas son identificados y registrados.

Las revisiones técnicas se llevarán a cabo para evaluar los productos del software o servicios y proporcionar evidencia de:

- Es completa.
- Cumple con las normas y especificaciones.
- El desarrollo, operación y mantenimiento se llevan a cabo de acuerdo con los planes, programas, normas y directrices.

4.3.4 Conclusiones

- Los casos de uso se utilizan en ámbitos de requisitos como diseño, ya que determinan los requisitos correctos y para luego ser utilizados para conducir el proceso de desarrollo.

- Se enfatiza la importancia de adoptar una aproximación iterativa e incremental en el desarrollo del software, esto conlleva un estudio de problemas en las partes del sistema que contienen riesgos, y a continuación se obtendrá una arquitectura estable completando partes más rutinarias en iteraciones sucesivas que de las cuales llevan a un incremento del progreso hasta la versión final.

- El desarrollo incremental ofrece la oportunidad de atender las necesidades de desarrollo del software que no pueden ser totalmente individualizados o identificados cuando se comienza el desarrollo. En efecto, como hemos visto para organizaciones emergentes, la necesidad continuada de reinventar significa que los requisitos del sistema son propensos a estar en un continuo estado de cambio. Para mantener el control de lo que es efectivamente un proceso de desarrollo es necesario un enfoque bien estructurado para la tarea de diseño.

- El objetivo de la calidad debe ser de idoneidad para un fin, aunque los criterios para determinar esto se logran tanto para problemas dependientes como dependientes en el dominio.

- El uso de la abstracción es una herramienta importante para el diseñador, el cual hace que sea difícil en cualquier medida del producto directo durante el proceso de diseño.

- Las revisiones técnicas de diseño pueden proporcionar un medio valioso para obtener y utilizar el conocimiento del dominio y ayudar con el diseño en la evaluación del producto, así como tener conocimientos de los distintos métodos que pueden ayudar a evaluar el proceso de diseño.

- En cada proyecto se debe establecer cuáles son los requisitos de calidad a cumplir y decidir los más importantes. Pero para poder asegurar y evaluar la calidad del software, esta se debe poder medir.

4.3.5 Descripción del diseño información de contenido

4.3.5.1 Introducción

El contenido requiere de un SDD lo siguiente:

- Identificación de la SDD.
- Identificadas las partes interesadas de diseño.
- Detectar problema de diseño.
- Seleccionar puntos de vista de diseño, cada uno con definiciones de tipo de sus elementos de diseño y lenguaje de diseño.
- Diseño de vistas.
- Diseño de superposiciones.
- Diseño de razón.

Lo anteriormente mencionado, se describe con detalle en el resto de esta cláusula.

4.3.5.2 SDD de identificación

El SDD deberá incluir la información descriptiva mediante lo siguiente:

- Fecha de emisión y el estado.
- Alcance.
- Organización de emisión.
- Autoría.
- Contexto.
- Uno o más lenguaje de diseño para cada punto de vista de diseño utilizados.

4.3.5.3 Partes interesadas del diseño y sus consideraciones

El SDD deberá identificar a los actores de diseño para el tema de diseño.
El SDD deberá identificar los problemas de diseño de cada actor de diseño identificado.
El SDD deberá incluir la preocupación de diseño identificado.

- Descripción base de datos

Se incluyen los siguientes aspectos:

1. Identificación e información general de la base de datos.
2. El diseño de base de datos proporciona:
   - Descripción de los niveles aplicables tanto lógicos como físicos.

3. Referencias de las descripciones del diseño del software para acceder a la base de datos.
4. Justificación para el diseño de la base de datos.
5. Decisión de diseño referente a la actividad a desarrollar desde la perspectiva de usuario como el adecuado cumplimiento de sus funcionalidades como de los requisitos.

- Descripción detallada del diseño de la base de datos

  La descripción detallada del diseño de la base de datos incluye elementos utilizados para acceder y por consiguiente la manipulación de los datos para obtener una mejor visibilidad en el diseño y una información de la gestión de la base de datos y elementos que están implicados en la base de datos el cual abarca los siguientes aspectos:

  1. Restricciones, limitaciones, características en el diseño de los elementos.
  2. Distintos tipos de errores que perjudican a la base de datos así como el control de los errores.
  3. Trazabilidad en la base de datos así como los elementos que están relacionados.
  4. Métodos de acceso en la base de datos.
  5. Entidades de los datos así como sus correspondientes relaciones.
  6. Restricciones de seguridad e integridad.
  7. Requisitos específicos de retención de datos.
  8. Tamaño de los elementos de datos.

- Descripción de diseño de alto nivel

  Refinamiento de la vista conceptual del sistema, determinar en el proceso funciones internas, descomponer funciones de alto nivel en subfunciones, además de los datos locales así como de su correspondiente almacenamiento. Las relaciones e interconexiones entre las distintas funciones. Se incluyen los siguientes aspectos:

  1. Definición de los distintos punto de vista para obtener la documentación en los distintos procedimientos: creación, interpretación, analizar y evaluación de los datos de la arquitectura.
  2. Cada punto de vista en la fase arquitectónica es una representación en sí de todo el sistema desde la perspectiva con respecto de uno a más problemas del sistema.
  3. Identificación de las partes interesadas y las consideraciones relacionadas con la arquitectura.
  4. Inspeccionar las propiedades y relaciones entre los elementos de una manera coherente.
  5. Justificar que los requisitos se cumplen y proporcionan un marco para el refinamiento del diseño.
  6. Definición de la interfaz existente o permanente para su desarrollo o su modificación.

7. Se debe definir y documentar los requisitos de las pruebas para la integración del software.
8. Se evaluará la arquitectura de los elementos del software así como la interfaz y la base de datos teniendo en cuenta que estará totalmente documentado en el cual abarcara: trazabilidad de los requisitos de los elementos del software, coherencia externa con los requisitos de los elementos del software, coherencia interna entre los distintos componentes en que se desarrolla el software, adaptación de los métodos de diseño y estándares a emplear.
9. Viabilidad del diseño detallado, operación y mantenimiento.

- Descripción de la interfaz

  Se especifica lo siguiente:

  1. Características de la interfaz de uno o varios sistemas, subsistemas.
  2. Dominios.
  3. Elementos tanto de hardware como de software.
  4. Manual de operaciones.
  5. Otros componentes del sistema.
  6. Sistemas o elementos de configuración de la interfaz a desarrollar.
  7. Normas y protocolos.
  8. Diagramas, control de errores.
  9. Definición de la interfaz existente o permanente para su desarrollo o su modificación.

- Descripción de diseño de bajo nivel

  Se incluye especificación de algoritmos, instrumentación de las funciones, especificación de algoritmo, estructura de datos que instrumenten el almacenamiento de los mismos, instrumentación de las funciones, interacción entre funciones y datos, se presenta características de uno a muchos sistemas, subsistemas, elementos, componentes e interfaces en el cual se especifica lo siguiente:

  1. Asignación de los requisitos específicos de elementos de software a componentes según sea necesario.
  2. Identificación de interfaces externas, unidad(es), componentes del software y otros.
  3. Ejecución, flujo de datos y control de flujo.
  4. Reutilización de los elementos.
  5. Control de errores.
  6. Especificación de los distintos protocolos.
  7. Separación de las entidades de diseño y características de las propiedades más relevantes y sus relaciones.

4.3.5.4 Opiniones de diseño

El propósito de una vista de diseño es hacer frente a problemas de diseño relacionados con el tema de diseño, para permitir que una parte interesada de diseño pueda centrarse en detalles de diseño desde una perspectiva específica y eficaz de los requisitos pertinentes.

Estudiaremos dos metodologías:

1. Metodología MERISE

   Es una metodología de análisis y diseño de sistemas de información aportando un plan de trabajo y técnicas de modelado para la concepción de aplicaciones para el área de gestión de empresas.

   Las características más importantes de MERISE son:

   - Resolución de problemas mediante los distintos niveles de abstracción, etapas de concepción y desarrollo definidas. Esto permite jerarquizar los problemas y aportar soluciones apoyándose en técnicas de modelado.

   - Modelo de tratamiento basado en redes de Petri, proporcionan una visión cronológica y completa de las relaciones entre procesos.

   - Coherencia de los sistemas resultantes, obtenida en la construcción de modelos de acuerdo con la empresa y el dominio

   Es un método de enfoque sistemático principalmente orientado a datos a través de las siguientes etapas.

   - Estudio previo: Define de manera global las soluciones conceptuales, organizativas y técnicas en relación con el dominio objeto de estudio.

   - Estudio detallado: Define explícitamente las especificaciones funcionales de la aplicación objeto de estudio.

   - Estudio técnico: Realiza la distribución de datos en ficheros físicos y el tratamiento en módulos de programas en función del sistema informático a utilizar.

   - Realización y puesta en marcha: Producción de los módulos de programación e implementación de los medios técnicos y organizativos necesarios así como formación del personal, lanzamiento de la aplicación y recepción definitiva por parte del usuario.

- Mantenimiento: Abarca el resto de la vida del sistema, hasta que sea sustituido por otro nuevo.

La metodología MERISE puede considerar una serie de puntos:

- Aproximación racional, abarca el ciclo de vida completo de un sistema de información asegurando su coherencia.
- Definición del método de trabajo.
- Utilización de los niveles de abstracción.
- Amplia utilización de diversas técnicas de modelado.
- Visión cronológica de los procesos.
- Amplio grado de difusión.
- Método en constante movimiento, lo que permite un alto grado de adaptación a las nuevas tecnologías informáticas.

2. La metodología SSADM

Las influencias de la organización sobre el diseño de la organización pueden ser consideradas como parte del proceso del método influenciado por un conjunto de factores no técnicos, derivados de la naturaleza y estructura de la organización mediante el método. Si bien estos factores no suelen afectar la parte de representación de manera tan directa, su influencia aún puede tener el efecto de ampliar o formalizarse de alguna manera. Para comprender la naturaleza de este método, la razón que está detrás de su adopción y uso deben ser considerados.

Los organismos internacionales, así como instituciones del gobierno central y local, son los principales clientes para los sistemas basados en software. Estos van desde sistemas de tiempo real para aplicaciones de control de stocks o de la fiscalía. Muchos de estos sistemas son muy grandes, son difíciles de especificar, los requisitos pueden cambiar con la tecnología, la legislación o la reorganización interna, y pueden ser producidos por los equipos en casa o por agencias de contratación externa.

Para ayudar con el control del aumento de grandes proyectos basados en software, necesarios en tales organizaciones, un crecimiento de respuesta en el énfasis es el uso de "estándar" métodos de análisis y, diseño.

El uso de un método estándar o la estrategia dentro de una organización tiene una serie de beneficios:

- Hay una interrupción mínima de un proyecto cuando se producen cambios de personal.

- Un cambio de director del proyecto no debe conducir a un cambio de dirección técnica.

- Normalización de la documentación permite una mejor gestión del mantenimiento.

- Existe la posibilidad de una mejor planificación, cálculo de costos y control de los proyectos.

- Basado en el uso de la experiencia pasada, que puede estar relacionada directamente con las prácticas actuales.

Hay, por supuesto, algunas de las características negativas de tales métodos para compensar estos positivos. Quizás el aspecto negativo más importante es que ellos tienden a ser excesivamente burocráticos, con el consiguiente riesgo de que las opciones creativas no puedan ser exploradas adecuadamente, y también que la dirección técnica en general se puede perder.

SSADM proporciona un conjunto de especificaciones y procedimientos para llevar a cabo sus correspondientes tareas de análisis y diseño de sistemas. No cubre planificación estratégica ni control de proyectos, el diseño físico llega hasta el máximo grado de detalle pero sin entrar en la construcción del código.

Estructura de la metodología SSADM:

- Estudio de viabilidad: El objetivo es llegar a establecer cuál puede ser el sistema de información que responda a los requisitos de negocio de la organización. Se recomienda hacer un estudio de viabilidad antes del análisis siempre que no se trate de proyectos con bajo riesgo.

- Análisis de requisitos: Se trata de definir el alcance del proyecto, establecer la integración de los componentes de la tecnología de la información con el resto de las necesidades de la organización, según los requisitos de los usuarios y conseguir una visión global de costes y beneficios que proporcionará el nuevo sistema.

- Especificación de requisitos: Proporciona una descripción detallada del futuro del sistema, estableciendo criterios de aceptación medibles que permitan llegar al diseño lógico con la garantía del acuerdo entre todas las partes implicadas.

- Especificación del sistema lógico: Se producen dos corrientes paralelas de actividad: definir opciones técnicas y la opción del sistema seleccionado se traducen en un conjunto de opciones de implementación técnica es decir, lenguajes, entornos de desarrollo, plataformas de producción.

- Diseño físico: El objetivo es especificar datos, procesos, entradas y salidas de acuerdo al entorno físico elegido, incorporando a todos ellos los estándares de la instalación

4.3.5.5 Punto de vista de diseño

Para cada punto de vista de diseño en un SDD, será exactamente un punto de vista de diseño que lo rige. Cada punto de vista de diseño deberá ser especificado por: nombre de punto de vista, consideraciones de diseño, elementos de diseño, los métodos de análisis, evaluación o técnicas de análisis.

Antes de explicar los distintos tipos de métodos de diseño desde un punto de vista vamos a destacar dos aspectos en tener en cuenta:

- **La importancia del emplear un método**

  - Proporciona un marco artificial para ayudar a pensar en un conjunto de elementos intrínsecamente invisibles ya que su importancia es abordada en gran parte en la representación de piezas de los métodos de diseño.

  - Las tareas cognitivas en la planificación y entender los grados de complejidad que acarrean, son utilizados mediante un enfoque sistemático y estructurado para el desarrollo y mantenimiento.

  - Necesidad de garantizar un comportamiento coherente y facilitar la integridad entre los diversos elementos implica un grado de normalización, por lo tanto este grado debe ser capaz de describir las estructuras involucradas.

  - Una de las características implícitas en los métodos de diseño es que al emplear distintos métodos de diseño dará lugar a emplear distintos estilos arquitectónicos asociado con ese método. Por lo tanto la larga vida de servicio que presta el software es preservar la integridad y la estructura de un sistema para desarrollar y mantener utilizando un método de diseño único.

  El uso de un método puede proporcionar un enfoque estructurado y sistemático para el proceso de desarrollo en el cual se establecen de la siguiente manera:

  - Cuestiones técnicas: Consisten en los aspectos relacionados con los problemas de diseño. Dependen en gran medida de una serie de factores externos tales como la estructura de la organización de desarrollo y la naturaleza del problema de diseño en sí.

  - Cuestiones de gestión.

  - Mecanismo de transferencia de conocimientos: Es una forma de llegar a una solución de diseño. Por lo tanto un método de diseño proporciona un descripción del procedimiento de cómo se dedicó a la tarea de conducir a una solución de diseño para un problema determinado es decir, un método describe las tareas que el

diseñador ha de desarrollar y el orden que se debe llevar a cabo ya que el diseño es un proceso creativo.

Un método no proporciona una orientación real exacta como cada tarea debe ser desempeñada para cualquier problema específico, los métodos de diseño son mucho más detallados y prescriptivos en los que surgen con otros dominios.

En resumen puede ayudar en la formulación y exploración en los mecanismos de transferencia de conocimientos utilizados para capturar las características esenciales del diseño. El conocimiento en los métodos puede ser un sustitutivo para el conocimiento del dominio cuando este es insuficiente e inexistente.

- El utilizar un método de diseño debe ayudar al diseñador para desarrollar un sistema que está estructurado de una manera consistente, en el cual es muy importante si el diseño está siendo realizado por un equipo de diseñadores que tendrá que verificar que todo cuadre correctamente, estableciendo un conjunto de normas, criterios y metas para utilizar el equipo.

- El emplear un método ayuda a la gestión cognitiva que participan en el diseño de un sistema. Reduce la probabilidad de errores en la estructuración lógica del sistema y garantiza que todos los factores involucrados en un problema queden encaminados y examinados por el diseñador(es).

- **Diseño de la máquina virtual (DMV)**

  Cada método de diseño proporciona una forma particular de “Diseño de máquina virtual (DMV)”, a su vez es el marco necesario en el diseñador para desarrollar un modelo de una solución.

  Cabe destacar que los diseñadores experimentados han desarrollado su propio diseño de la máquina virtual para determinar las clases de problemas. Aunque el concepto de un estilo arquitectónico forma uno de los componentes claves DMV, la determinación de la forma del conjunto de elementos de diseño, el entorno de tiempo de ejecución que se utilizan para la aplicación. Cabe destacar que DMV es más que un estilo arquitectónico.

  El concepto de DMV no suele ser aplicado al diseño de métodos. DMV proporciona una capa de abstracción por encima de la capa física. El sistema operativo proporciona una serie de máquinas virtuales que permite a los programadores acceder a los recursos de un ordenador en diferentes niveles de abstracción. Ejemplo, un fichero puede ser descrito como una secuencia de caracteres, secuencia de registros, o secuencia de bloques en un disco.

Los lenguajes de programación proporcionan máquinas virtuales entre ellos cabe destacar: el lenguaje java que tiene su propia máquina virtual o C++; el programador trabaja en su equipo sin necesidad de entender la arquitectura subyacente de los registros, buses, etc.

Como se ha mencionado anteriormente cada lenguaje de programación proporciona una maquina virtual cuya arquitectura y forma están especificadas por las características y semánticas del lenguaje de programación. Por consiguiente un método de diseño proporciona al usuario una máquina virtual que se puede usar para expresar sus ideas sobre las formas del programa en un nivel muy alto, debido a la semántica de las representaciones del modelo de diseño resultante se traduce en un menor nivel de abstracción en el diseñador y el programador juntos.

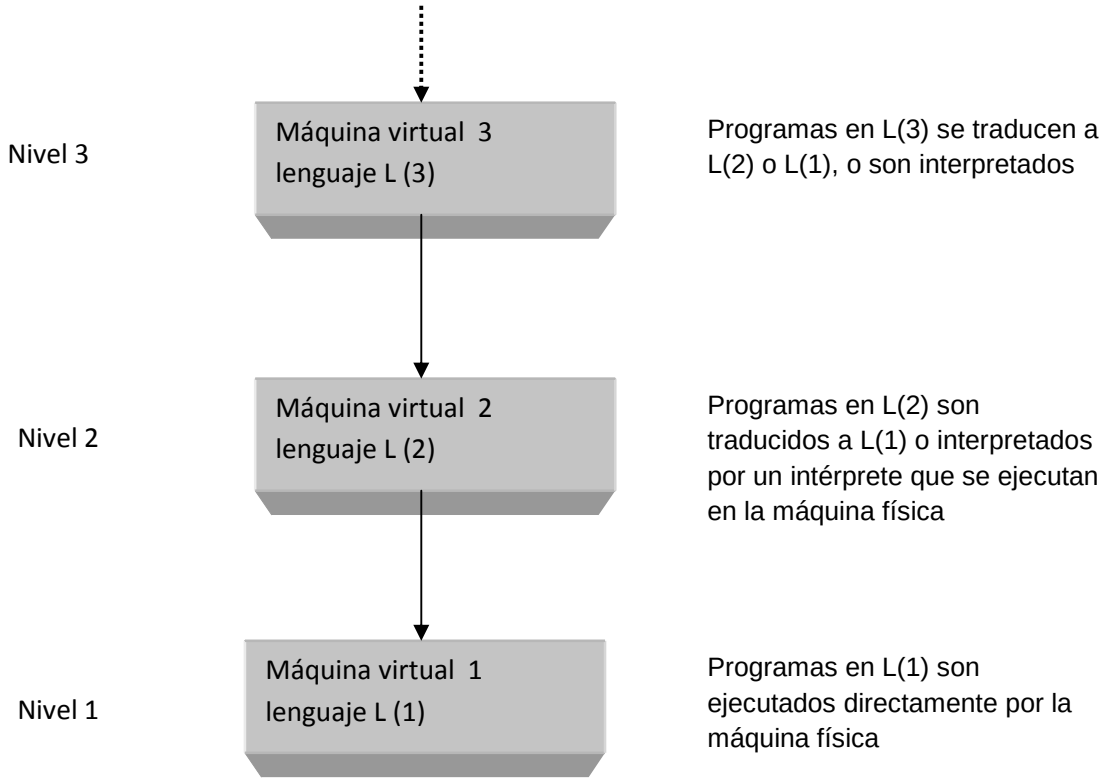


Figura 56.Uso de máquinas virtuales en el proceso de diseño. Adaptado de [David Budgen]

Esto implica dos niveles de DMV:

- Utilización para el diseño arquitectónico o lógico. El modelo de producción se traduce en un plan de diseño (diseño físico) que es otro nivel de DMV con características más semejantes que el DMV inicial y con estructuras mucho más cercanas a la posterior ejecución.
- Proceso necesario para traducir en un lenguaje de programación que puede ser traducido por la propia máquina y conduce a la ejecución del sistema en la maquina física.

El DMV se materializa en un método de diseño particular, establece factores como:

- Estilo arquitectónico implícito, forma de los elementos a emplear.
- Punto de vista utilizado en las diferentes fases del proceso de modelización.
- Relaciones entre cada uno de ellos que se establecen a través de la forma del proceso de diseño.
- Estrategia básica del método de diseño en sí ya que determina la elaboración de los modelos de diseño.

Lo anteriormente mencionado puede crear un conjunto de supuestos acerca de cómo generar la solución es decir (secuencial, paralela, centrada en los datos, en los objetos etc.). Que a su vez forma una parte primordial del marco desempeñado por el diseñador.

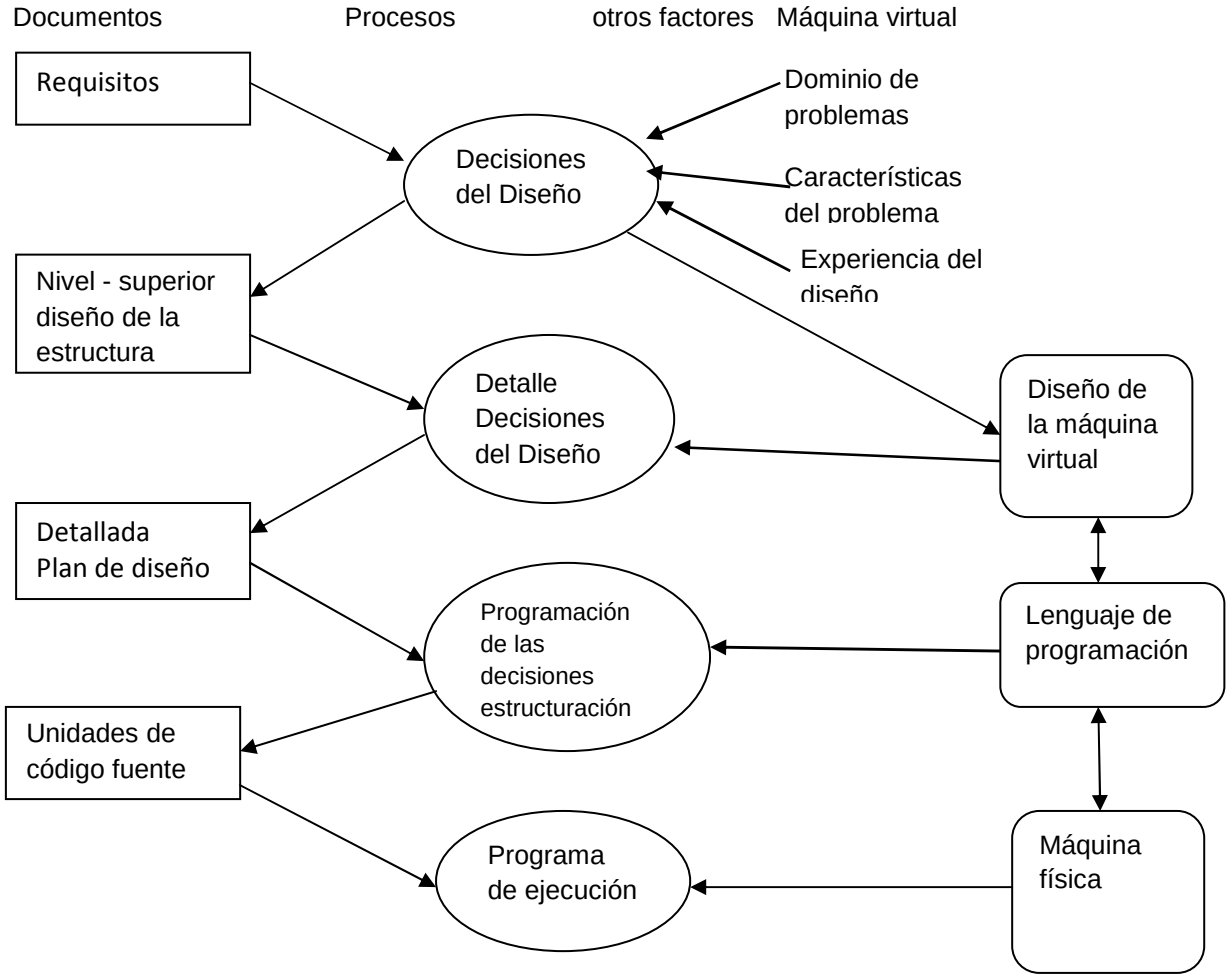


Figura 57.La relación entre el DVM y el nivel de máquina virtual utilizado en una computadora. (Cada método tiene una máquina virtual de diseño incorporados en su interior, ayudando a determinar aspectos tales como el conjunto de puntos de vista empleado, la estrategia, las hipótesis de arquitectura, etc.). Adaptado de [David Budgen]

A continuación explicaremos algunos métodos de diseño.

Se proporciona un resumen de los puntos de vista sobre los métodos de diseño de software que cada uno de estos métodos pueden proporcionar, y la lista de las características del método de diseño que se puede capturar a través de su uso.

Los tres componentes que componen un método de diseño:

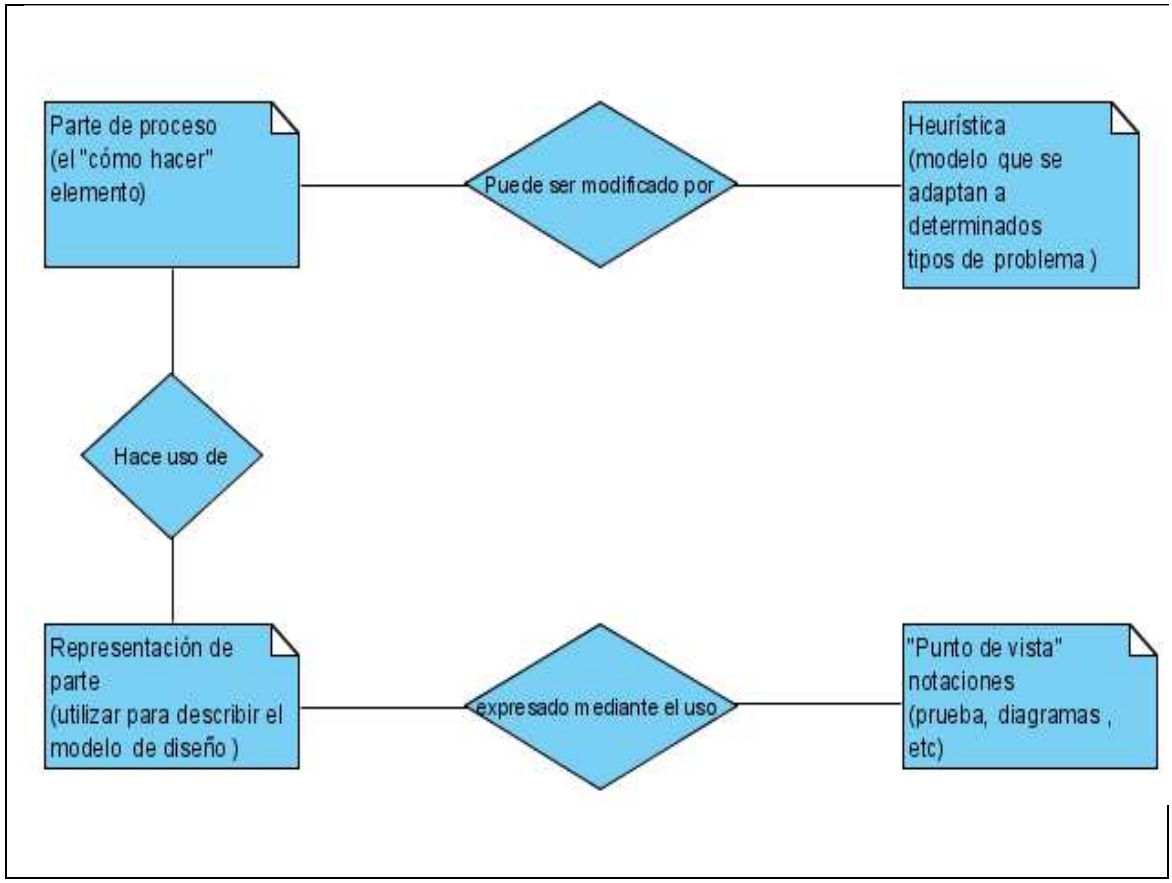


Figura 58.los tres componentes de los métodos de diseño. Adaptado de [David Budgen]

La parte del proceso de un método proporciona la representación procesal de las directrices sobre como los modelos se acercan a la forma de la parte del proceso.

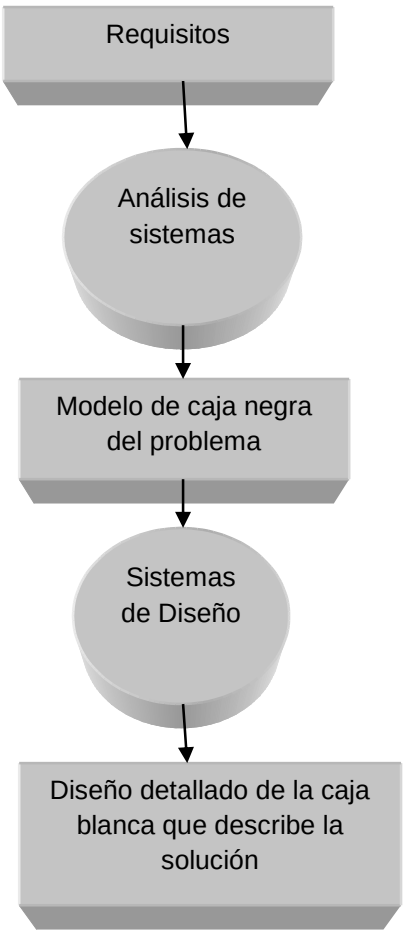


Figura 59.Modelo general del proceso del diseño de software. Adaptado de [David Budgen]

En la figura 59 muestra un modelo de proceso del diseño de software de una forma simbólica que se utilizará para describir el método de diseño. A continuación se ampliará con más detalle la figura 60.

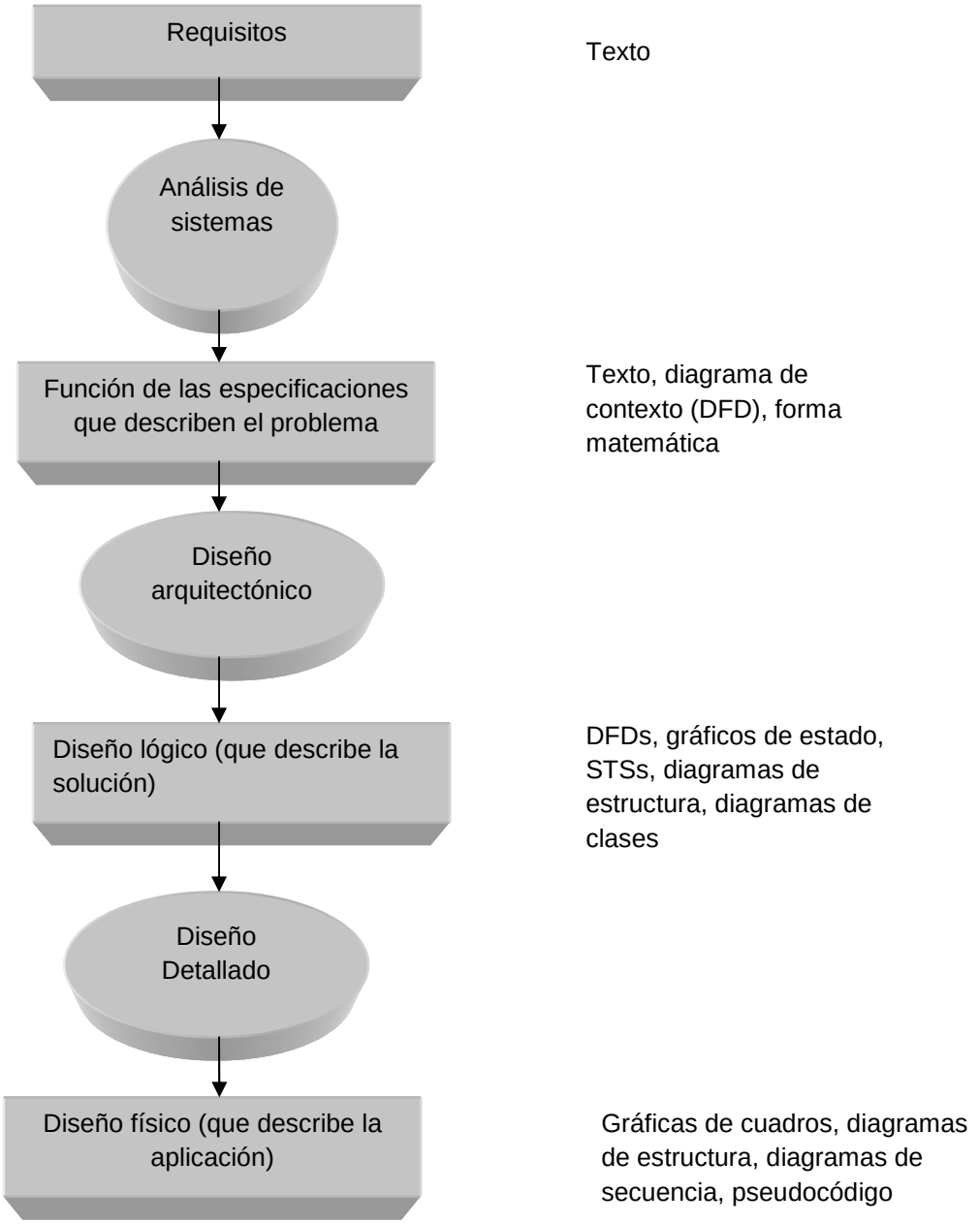


Figura 60.Vista ampliada del modelo de procedimiento del proceso del diseño de software. Adaptado de [David Budgen]

En la figura 60 puede observarse más ampliada con respecto a la figura 59. Se describe la estrategia global ya que será conducido por las necesidades de un problema específico y la experiencia del diseñador.

Muchos factores han influenciado en el desarrollo de métodos de diseño y por lo tanto es muy complejo de clasificar de una manera sistemática.

Nombre	Descripción
Métodos descomposicionales	Arriba hacia abajo es una visión del proceso de diseño, modelado de diseño a través de un proceso de subdivisión
Métodos de composición	Se construye a partir de la identificación de las entidades de alguna forma
Métodos de organización	La estructura del proceso del diseño debe estar relacionado con los requisitos que deben cumplir y con los requisitos no funcionales que se basan en la forma de la organización
Métodos plantillas	Dominio específico del problema, proporciona una clase de problemas que pueden realizarse a través de una estrategia estándar

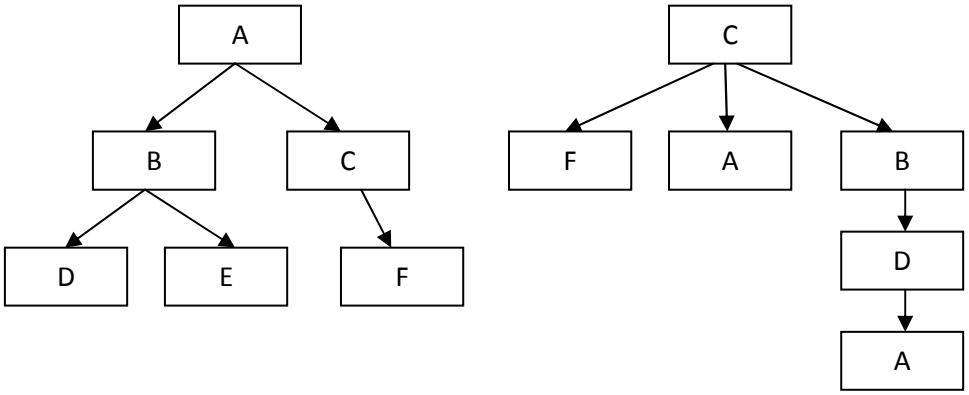
Tabla 149.Descripción de los métodos de diseño a emplear

- Diseño por la descomposición de arriba hacia abajo

  La mayoría de los lenguajes de programación tienen un mecanismo importantes para la descripción orientada a la estructuración de un programa mediante el uso de subprogramas, en el cual rara vez proporcionan los medios de crear y utilizar estructuras de datos muy complejas.

  Prestando atención a este énfasis entre las formas de aplicación, una estrategia de diseño es que se subdivide la tarea principal de un programa en tareas más pequeñas, con esta subdivisión se continúa hasta que las subtareas resultantes sean suficientemente elementales para poner en ejecución subprogramas.

  El éxito de este enfoque de encontrar una solución depende en gran medida de la forma en que se describe el problema original. Ya que el diseñador se basa en la elección inicial de las subtareas. Se puede argumentar que este enfoque es por lo tanto intrínsecamente inestable, ya que puede diferir en las diferencias de descomposición de una tarea y puede conllevar soluciones que son funcionalmente equivalentes, pero estructuralmente muy diferente.



Solución (a) Solución (b)

Figura 61.Descomposición de solución de un problema simple. Adaptado de [David Budgen]

Esta inestabilidad se debe que al seguir una estrategia de arriba hacia abajo, las decisiones importantes sobre la estructura básica se efectuaron al inicio del diseño de proceso, por lo tanto lo realizado se propaga a través de los siguientes pasos.

Esto puede conducir a problemas en una fase posterior de diseño e incluso a la necesidad de rediseñar totalmente el sistema. En resumen siempre que se utilice la estrategia de arriba hacia abajo es importante explorar las opciones de diseño cada vez que se requiera en cada etapa de descomposición.

Además la utilización de la estrategia de arriba hacia abajo implica el problema de la duplicación esto es debido a la secuencia de refinamientos. Existe la posibilidad de la duplicación total o parcial de la funcionalidad cuando las operaciones de bajo nivel se definen, el alcance es mayor si el diseño es responsabilidad de más de una persona. Por lo tanto el emplear el proceso de refinamiento implica la necesidad de comprobación y resolver cualquier duplicación.

- Métodos de composición de diseño

  Es una estrategia de arriba hacia abajo para el diseño. Está centrada en la identificación de las operaciones que deben llevarse a cabo por el sistema.

  Los resultados del modelo del problema se basan en su totalidad desde el punto de vista funcional, aunque el refinamiento del modelo puede realizarse desde otros puntos de vista. El reverso de este enfoque consiste en emplear la estrategia de composición para la construcción de modelos del diseñador, un modelo del problema es construido por las descripciones en desarrollo de un conjunto de entidades u objetos reconocidos en el propio problema y a su vez con la descripción de las relaciones que se conectan con las entidades.

Las entidades utilizadas pueden variar con el método de igual manera los puntos de vista elegido para las características de ellos. Las entidades normalmente se identifican mediante el análisis de las características iniciales del problema.

Los métodos de composición son más complejos que los métodos que se basan en la estrategia de arriba hacia abajo y por tanto requiere una mayor disciplina en su utilización por lo tanto se argumenta lo siguiente.

- Son más estables ya que su utilización conduce a soluciones similares para un problema, indistintamente del usuario ya que la estrategia de diseño relaciona la estructura de la solución a la estructura del problema.

- Proporciona mejores condiciones en un proceso de verificación entre el diseño y la especificación inicial.

- Proporciona múltiples puntos de vista para su realización en el cual conlleva a explotar el uso de conceptos primordiales como es el diseño modular y ocultación de la información.

  Esto surge en la elaboración de un modelo de diseño con ámbito de aplicación a los elementos considerando las distintas relaciones entre ellos incluyendo la estructura de datos compartida. Este es un proceso difícil de conseguir a través de la utilización de una estrategia de arriba hacia abajo

- Método de diseño de la organización

  Un método de diseño de la organización puede ser considerado en que la parte del proceso del método está influenciado por un conjunto de factores no técnicos derivados de la estructura de la organización mediante el método.

  Estos factores no tienen por qué afectar a la parte de representación de una manera tan directa. Los organismos internacionales, instituciones del gobierno central y local son los principales usuarios para sistemas basados en software.

  Estos sistemas son muy complejos, difíciles de especificar los cambios de los requisitos cambian muy fácilmente mediante las tecnologías, legislación y la reorganización interna. Para el crecimiento de grandes proyectos basados en software necesarios para las organizaciones es utilizar “estándar” métodos de análisis y diseño en el cual presenta una serie de beneficios:

  - Interrupción mínima de un proyecto cuando se producen cambios de personal.

- Cambio de director del proyecto no tiene por qué conducir a cambios de dirección técnica.
- Normalización en la documentación permite una mejor gestión de mantenimiento.
- Posibilidad de una adecuada planificación, cálculo de costes y control de proyectos.
- Una característica negativa más relevante tiende a ser excesivamente burocráticos, con el riesgo que las opciones de creación ser exploradas adecuadamente y también que la dirección técnica en general se puede perder.

En términos de estrategia que intervienen en su utilización, los métodos de organización difieren considerablemente de acuerdo a su ámbito de aplicación. La organización la podemos catalogar especialmente independiente de la estrategia de modo que un método puede ser de arriba hacia abajo y de organización o de composición o de organización.

4.3.5.6 Elementos de diseño

Un elemento de diseño es cualquier objeto que ocurre en una vista de diseño. Un elemento de diseño puede ser cualquiera de los siguientes subcasos: entidad de diseño, la relación de diseño, atributos de diseño, o la restricción de diseño.

4.3.5.6.1 Entidades de diseño

Las entidades de diseño capturan los elementos clave de un diseño de software. Cada entidad de diseño deberá tener un nombre, un tipo, y el propósito.

Para ello se contemplan los siguientes aspectos a tener en cuenta:

- Sistemas: Se emplean en muchos ámbitos distintos. Se emplean para designar un concepto, herramienta genérica que se puede emplear para explicar o analizar mejor como es ó que ocurre en una determinada área.

- Marcos: Un marco de plantilla es un conjunto de clases que constituyen una aplicación incompleta y genérica.

- Patrones: Describen un problema que ocurre una y otra vez en nuestro entorno, así como la solución a ese problema. Idea de diseño ampliamente probada y documentada.

- Plantillas genéricas: Especificación de una clase en la cual permanecen sin concretar algunos aspectos de sus operaciones y atributos en forma de parámetros a los que se debe dar un valor.

- Componentes : Los podemos definir de la siguiente manera:

  - Son elementos del software que se ajustan a un modelo de componentes y pueden ser desplegados de forma independiente y compuestos sin modificaciones de acuerdo a un estándar de composición.
    - Conjunto de clases que colaboran para llevar a cabo una función concreta. Exteriormente se ve como una clase que implementa una interfaz determinada.
    - Un objeto físico que existe en tiempo de desarrollo, de compilación ó de ejecución posee identidad propia e interfaz.

- Clases: Conjunto de objetos que tiene los mismos atributos y operaciones: una clase puede ser:

  - Clase Abstracta: Superclase de la cual no se pueden instanciar objetos porque cada uno de sus objetos debe pertenecer a alguna de sus subclases.
  - Clase Asociativa: Por el simple hecho de poseer atributos y operaciones propias, llega a ser una clase.
  - Clase control: Clase no persistente que implementa todos los algoritmos de un caso de uso o parte de ellos.
  - Clase entidades: clase del dominio del software
  - Clase frontera: Clase que implementa una parte de la interfaz de usuario en el ámbito del análisis.
  - Clase de utilidad: Recursos para agrupar procesos o datos en forma de una clase que no puede tener objetos.
  - Clase diferida: Clase abstracta que tiene alguna operación abstracta.
  - Clase persistente: Clase que puede tener objetos que se deban hacer persistentes.
  - Clase temporal: Clase que no es persistente, cuyos objetos no se pueden hacer persistentes.
  - Clase terminal: Clases cuyas subclases no se permitan que se definan.
  - Clasificador: Concepto que comprende las clases, las interfaces y los tipos de datos.

- Almacenes de datos: Se definen como entidades repetitivas de datos o grupos de datos. El analista o usuario selecciona uno o más datos para organizar la colección de entradas en un almacén que se denomina identificador.

- Descomposición modular: Se trata de aplicar el concepto de modularidad y obtener una división del sistema en partes o módulos. Todas las técnicas de diseño están de acuerdo en la necesidad de realizar una descomposición modular del sistema como actividad fundamental del diseño.

- Unidad de programa: es una unidad de código fuente que es desarrollada o mantenida por una persona; esa persona es la responsable de la unidad. En un sistema bien diseñado una unidad de programas es un subprograma o grupo de subprogramas que cumplen una función bien definida o forman un subsistema bien definido.

4.3.5.6.2 Atributos de diseño

A los nombres de atributos de diseño una característica o propiedad de un elemento de diseño (que puede ser una entidad de diseño, restricción de diseño, o una relación de diseño) y ofrece una exposición de hechos de ese elemento de diseño.

Para cada uno de los subapartados de las secciones siguientes se detalla el concepto de atributo dependiendo del diseño de las perspectiva, a continuación vamos a explicar el atributo con respecto a la calidad.

La característica de casi todos los buenos diseños, en cualquier ámbito de la actividad que se producen, es una simplicidad básica. Albert Einstein dice "*la solución debe ser lo más simple posible pero no lo más sencillo*". Se puede lograr la simplificación mediante la abstracción, pero es necesario el uso de la abstracción que conserva atributos. La simplicidad no puede ser fácilmente evaluada, se pueden buscar por lo menos características inversas de la complejidad.

Varios de estos miden las diferentes formas de complejidad que se han desarrollado para el uso del software:

- Complejidad de control de flujo: Número de posibles caminos de control durante la ejecución de una unidad de programa.
- Complejidad de la estructura en términos del flujo de la información en todo el sistema
- Complejidad de la compresión: Medida por el número de identificadores de los diferentes operadores.
- Complejidad de la estructura en términos relacionados entre los elementos del sistema, principalmente orientada a objetos.

En cuanto a diseño de calidad, la simplicidad está claramente relacionada con la mantenibilidad y la capacidad de prueba, así como la eficiencia y fiabilidad posible.

Visto en general, y también desde la perspectiva del software, encontrar un plan adecuado de la modularidad de aplicar la solución de un problema dado proporciona por lo menos los siguientes beneficios:

- Los módulos son fáciles de reemplazar.
- Cada módulo capta un aspecto de un problema, por lo que ayuda a la comprensión y al mantenimiento.
- Un módulo bien estructurado puede ser reutilizado por otro problema.

En términos de las propiedades de diseño, el uso exitoso de la modularidad debe estar relacionado con conceptos de calidad como son: mantenimiento, capacidad de prueba, usabilidad y fiabilidad.

Dos medidas de calidad útiles que han sido utilizadas con el propósito de evaluar el grado de estructuración modular de software son: acoplamiento y cohesión. Estos términos están basados en los problemas que surgen en el desarrollo de grandes sistemas, y se utilizan para la identidad de la complejidad de un sistema en términos de la forma e interdependencia de los módulos que lo componen.

La unidad básica de la modularidad es el subprograma, pero los conceptos siguen siendo válidos para tales formas modulares como "paquete" de Ada, el Modula-2 "módulo" y "clase" de Java. Aunque los términos de acoplamiento y cohesión se estudian con más detalles en la sección siguiente, explicaremos brevemente estos dos conceptos:

- Acoplamiento: El acoplamiento es una medida de la conectividad de intermódulos, y se ocupa de identificar las formas de conexión que existe entre los módulos.
- Cohesión: Proporciona una medida del grado en que los componentes de un módulo pueden ser considerados como funciones relacionadas. El módulo ideal aquel en el que todos los componentes se pueden considerar como el único presente con un propósito.

En cuanto a los factores de calidad del sistema están relacionados con la confiabilidad y mantenibilidad. En términos de tiempo de procesador y el uso de memoria, es apto para entrar en conflicto con las ideas de eficiencia, ya que el uso de una jerarquía de los procedimientos de acceso se puede llevar a un menor rendimiento en tiempo de ejecución.

Las formas detalladas en las que estos atributos se manifiestan en los sistemas de software son algo difícil de alcanzar, a pesar de los conceptos involucrados son en general bastante manejables. Una ayuda útil para decidir el alcance de su presencia en un diseño es revisar las características que indican que un diseño carece de estas cualidades.

Esto implica la identificación de un conjunto adecuado de las funciones correspondientes a los atributos. La presencia de cualquiera de estas funciones se especifica a continuación:

- Identificar los atributos que son los inversos directos de los descritos anteriormente es un problema significativo. Sin embargo, las siguientes características indeseables de un diseño pueden considerarse una consecuencia de la presencia de las propiedades inversas.

- Después de haber muchas copias y de difundir información del estado en todo el sistema. Esto complica la actualización de la información, y puede dar lugar a incoherencias en el comportamiento del sistema.

- Uso de interfaces que son demasiado complejas. En términos de programación por lo general son caracterizados por los procedimientos que tienen demasiados parámetros, algunos de los cuales ni siquiera pueden ser necesarios. En términos de diseño, son más propensos a ser identificados por el grado de flujo de la información asociada a ellos.

- El uso de estructuras de control excesivamente complejas. Esto puede indicar módulos de diseño donde el diseñador ha elegido interfaces adecuadas y estructuras de datos.

- El programa de unidades no está bien enfocado en cuanto a su fin, y bien puede ser parametrizado para su uso en diferentes tareas.

Estas características se han expresado en términos de características de implementación, ya que son más familiares y se cuantifican más fácilmente. Sin embargo, puede ser igualmente evaluado en la descripción del diseño, aunque la extracción de la información es menos fácil de lograr a través de cualquier forma de análisis.

4.3.5.6.2.1Nombre del atributo

El nombre del elemento. Cada elemento de diseño tendrá un nombre de referencia inequívoca. Los nombres de los elementos pueden ser seleccionados para caracterizar su naturaleza. Esto simplificará la referencia y seguimiento, además de proporcionar su identificación.

Un mecanismo de abstracción en un lenguaje de programación es el uso de nombres, es decir identificadores para denotar entidades o constructores del lenguaje. En la mayoría de los lenguajes, las variables, procedimientos, constructores, constantes pueden tener nombres asignados por el programador. Un paso fundamental de la descripción de la semántica de un lenguaje es describir las reglas convencionales que determinan el significado de cada uno de los nombres utilizados en el programa.

El significado de un nombre queda determinado por las propiedades o atributos asociados con el mismo. Los atributos no deben confundirse con palabras claves, podemos hacer referencia a un atributo mediante la palabra clave que lo establece. Las declaraciones no son los únicos constructores del lenguaje que pueden asociar atributos a los nombres. En algunos lenguajes los constructores hacen que los valores sean vinculados con nombres.

4.3.5.6.2.2 Tipo atributo

Una descripción del tipo de elemento. El atributo de tipo deberá describir la naturaleza del elemento.

A continuación veremos una serie de casos de los atributos:

- Atributo cuyo valor es compartido por diferentes objetos. Serán atributos de clase.

- Atributos no aplicables a todos los objetos de la clase. Se puede crear una superclase que tenga sólo los atributos que sean aplicables a todos los objetos y después una subclase o más atributos que sean aplicables a todos los objetos respectivamente.

- Atributos con valores repetitivos: A veces es conveniente definirlos como clase aparte con una asociación n a 1 con la primera, sobre todo si son atributos compuestos o pares de atributos con la misma cardinalidad.

- Atributos derivados: Sólo se incluirán si figuran en salidas descritas en los casos de uso, mientras que los recursos pueden reducir la duración de los procesos a cambio de utilizar más espacio en memoria o en la base de datos no se introducirán hasta el diseño, que es cuando se tiene en cuenta este punto de vista.

Especificación de los atributos de las clases de entidades

Los atributos de las clases de entidades son generalmente datos mencionados de forma explícita en los casos de uso. Lo que significa que los usuarios lo utilizan directamente. Puesto que un modelo de análisis conviene que sea independiente del lenguaje de programación concreto.

Los tipos de atributos serán tipos abstractos y no tipos soportados por un lenguaje de programación concreto. Conviene que se establezcan cuáles son estos tipos. Por la misma razón los nombres de los atributos no deben ser sometidos necesariamente a las restricciones de los nombres de ningún lenguaje de programación concreto, pero para evitar cambiar los nombres al llegar al diseño, conviene que los nombres de atributo tengan un formato compatible.

Los tipos abstractos de datos permiten crear nuevos tipos de datos además de los predefinidos en el lenguaje de implementación. Todos los datos concretos que se manejan en un programa deben aparecer como constantes o variables de algún tipo, sea predefinido o definido como tipo abstracto. Para operar con un dato en particular se invocará alguna de las operaciones definidas sobre su tipo, indicando a que dato en particular queremos aplicarla.

4.3.5.6.2.3Propósito atributo

Una descripción de por qué existe el elemento. El objetivo del atributo deberá proporcionar la justificación para la creación del elemento.

Un atributo puede clasificarse de acuerdo con el tiempo en que está calculando y vinculando con un nombre durante el proceso de traducción/ejecución. Esto se conoce como tiempo de ligadura del atributo. Los tiempos de ligadura pueden clasificarse en dos clases generales: ligadura estática y ligadura dinámica. La ligadura estática tiene lugar antes de la ejecución. Un atributo que esté vinculado estáticamente es un atributo estático, y por consiguiente un atributo que esté vinculado dinámicamente es un atributo dinámico.

Los tiempos de ligadura pueden refinarse aún más en subcategorías de ligadura estática y dinámica. Un atributo estático puede vincularse durante el análisis gramatical o durante el análisis semántico es decir tiempo de traducción, durante el encadenamiento del programa con la biblioteca es decir tiempo de ligado o durante la carga del programa para su ejecución es decir tiempo de carga.

Los nombres pueden vincularse con atributos aun antes del tiempo de traducción. Los identificadores predefinido tienen su significado y por tanto sus atributos especificados por la definición del lenguaje.

4.3.5.6.2.4Atributo autor

Identificación de diseñador.

4.3.5.6.3 Relaciones de diseño

Correspondencia entre dos o más entidades de diseño. Ofrece una exposición de hechos sobre las entidades de diseño.

En este apartado se especifica con más detalle en la sección siguiente, donde se contemplan: los métodos estructurados de diseño se construyen alrededor de las relaciones de diseño, métodos de diseño orientado a objetos utilizan las relaciones de diseño, como la encapsulación, la generalización, especialización, la composición, la agregación, la realización y creación de instancias.

4.3.5.6.4 Restricciones de diseño

Una restricción de diseño es un elemento de una vista de diseño o restricción impuesta por un elemento de diseño (el origen) a otro elemento de diseño (el objetivo), que puede ser una entidad de diseño, atributos de diseño, o una relación de diseño.

En el diseño del software, se puede identificar fácilmente muchas de las restricciones impuestas a la forma del producto. Algunos pueden ser determinados por el posible entorno de ejecución es decir, organización de las estructuras de los archivos, el aspecto de la interfaz de usuario etc., mientras que otros se refieren a la necesidad de adaptar los convenios requeridos por un estilo arquitectónico elegido.

Donde también se pretende que el sistema final será construido mediante la reutilización de componentes de software, esto puede llevar a restricciones adicionales aceptables, estilos arquitectónicos que los componentes pueden poseer, así como la atribución de funciones entre los componentes.

Una de las limitaciones más importantes en la tarea de diseño y la forma del diseño en sí mismo es el de la posible forma de implementación. Durante muchos años el enfoque utilizado por la mayoría de los proyectos de desarrollo de software ha sido la de comprar el hardware y seleccionar las instalaciones principales de software es decir, sistema operativo, lenguaje de programación antes de dar inicio a la tarea de diseño en sí. La elección del lenguaje de programación es aún más probable que sea determinado por factores externos como las aptitudes del programador y el conocimiento que por las características del problema en sí.

Las formas del lenguaje imperativo de programación tales como: Ada, C, C + +, Java y lenguajes scripting como JavaScript, etc. se han mantenido en la herramienta dominante en la producción de software. De hecho, el uso de las construcciones del lenguaje imperativo está implícito en casi todos los métodos de diseño disponibles en la actualidad.

Las limitaciones en el proceso de diseño son más difíciles de identificar. Pueden estar relacionadas con las cuestiones de diseño y el conocimiento es decir, la experiencia con un método particular, o con una necesidad de adaptarse a un determinado "estilo arquitectónico" de diseño. En algunos casos, la limitación del producto lleva a una restricción en el proceso, por ejemplo, una forma de salida debe ser coherente con lo que se produce a partir de una estrategia de diseño particular.

Cualquiera que sea la forma que se adopte, las restricciones pueden ser consideradas como un conjunto de límites en un espacio de la solución. Si el proceso de diseño no converge a una solución para un problema dado, los efectos de las restricciones pueden limitar la cantidad de divergencia posible que sea aceptable en circunstancias particulares.

Muchas limitaciones serán identificadas en los documentos de especificación inicial, aunque esto no necesariamente sea cierto para aquellos que están relacionados con las expectativas de la reutilización. Cualquiera que sea su fuente, las restricciones limitan el espacio de la solución global mediante la limitación de una variedad de opciones disponibles para el diseñador. Puede existir el riesgo, de las inconsistencias en la especificación de hecho puede hacer que sea imposible encontrar una solución.

Algunas restricciones son problemas específicos tales como el nivel de conocimientos de los usuarios, mientras que otros son más específicos como es la solución del estilo arquitectónico.

Un problema específico de las limitaciones es que no pueden ser fácilmente incorporadas en cualquier proceso de diseño formal como es un método de diseño, su influencia es necesario considerarla a cada paso. Una forma de identificar y rastrear las limitaciones es mediante el uso de las revisiones de diseño regular, en el cual es muy importante el rol del aseguramiento de la auditoría.

4.3.5.7 Plantilla de diseño

Una plantilla de diseño se utiliza para presentar información adicional con respecto a una vista de diseño ya definido.

Introduciremos el concepto de patrón de diseño: se basa en la obra del arquitecto Christopher Alexander, y lo describe mediante las siguientes palabras: *"Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno y a continuación se describe el núcleo de la solución a ese problema de tal manera que se puede usar esa solución más de un millón de veces, sin tener que hacerlo de la misma manera dos veces."*

Este concepto se ha introducido en el software especialmente para el software orientado a objetos. Un patrón es una solución específica a un problema que a su vez dependa de un problema mayor, en cuanto a la trasferencia de conocimientos se transmite información sobre un problema en particular que el diseñador tendrá que afrontar junto a una estrategia para resolver el problema, así como las consecuencias que tendría que tener en cuenta al desarrollarlo.

Los patrones son un enfoque que podría optar para organizar la transferencia de conocimientos de diseño, de hecho es un mecanismo muy eficaz que impone limitaciones para tener éxito, los patrones deben ser fácilmente accesibles y reconocibles. El concepto de patrón es un nivel útil de la reutilización de los conocimientos de diseño. Siempre tenemos que tener presente si el uso de patrones nos guiará a buenas soluciones ya que su utilización podría producir soluciones que tienen un estilo coherente.

La reutilización de la experiencia de las personas cuando se enfrentan a problemas es a través de alguna manera la "plantilla de diseño". Esta plantilla de diseño proporciona una abstracción de los conocimientos que otros han

adquirido acerca de cómo estructurar las soluciones para un determinado tipo de problema que se está repitiendo.

La plantilla de diseño para que sea eficaz como medio de intercambio de experiencias en lugar de facilitar la reutilización de un solo diseñador necesita el apoyo de algún tipo de mecanismo que se pueda usar para describir y registrar la plantilla. La plantilla de diseño no es una especificación en detalle se puede observar como una descripción del conocimiento y la experiencia es una solución a un problema.

En el apartado 4.3.7.7 vamos a estudiar el uso de diseño de patrones orientados a objetos considerando las consecuencias de sus éxitos y finalmente lo que queda por hacer para que este enfoque sea aprobado para su utilización con otras formas de estilo arquitectónico, lo cual describe clases y objetos que están relacionados para solventar problemas de diseño en un determinado contexto. El patrón de diseño identifica y abstrae los aspectos importantes de una estructura de diseño que crea y utiliza un diseño orientado a objetos reutilizables. Por lo tanto cada patrón de diseño tiene como tarea centrarse en un problema en particular describiendo cuando hacer uso de ello, prestando atención a otras restricciones de diseño así como las consecuencias, ventajas e inconvenientes para su utilización.

A continuación explicaremos como los patrones de diseño están codificados, y se describirá mediante una plantilla de descripción del gráfico, se incluyen los siguientes aspectos:

- Nombre utilizado para describir la esencia del patrón.
- Problema: Es el problema de diseño que el patrón tiene como objetivo a abordar.
- Solución: Es la manera en como el patrón aborda el problema
- Ejemplo: Proporciona un ejemplo del mundo real de un problema de diseño en particular y muestra cómo el patrón lo trata.
- Aplicación: Es una descripción de la situación en la que podría ser un patrón empleado, e incluye todas las sugerencias para el reconocimiento de dicha situación.
- Estructura: Proporciona una descripción detallada de la solución utilizando notaciones gráficas empleadas para describirlo en términos tanto de comportamiento y los puntos de vista de la construcción.
- Implementación: Proporciona un conjunto de pautas para implementar el patrón, y tomar nota sobre cualquier dificultad que pudiera surgir.
- Usos conocidos: Es un conjunto de ejemplos que se recogen en los sistemas existentes, lo recomendado sería utilizar más de un dominio.
- Patrones relacionados: Identifica patrones que abordan problemas semejantes o se complementan de alguna manera.
- Consecuencias: Resultados de aplicar el patrón, así como las limitaciones de ningún uso, podría imponerse en términos de estructura y comportamientos del sistema.

4.3.5.8 Justificación de diseño

Razón de ser del diseño capta el razonamiento del diseñador que llevó al sistema tal como fue diseñado, y los motivos de dichas decisiones.

En muchos casos un buen diseño debe ser aquel que se pueda realizar utilizando una gran variedad de tecnologías, un modelo de solución no tiene por qué suponer una posible aplicación pero la evidencia no sistematiza como lo es pero se sugiere lo que puede suponer, la posible forma de ejecución crea algunas limitaciones al diseñador especialmente referentes a la optimización y reutilización.

Hemos estudiado algunos modelos acerca del conocimiento y comprensión del diseño de sistemas basados en el software. Este conocimiento puede ser incompleto incluso puede ser difícil de interpretar o codificar e igualmente resulta bastante complicado expresarlo de una manera clara. Este conocimiento como lo podemos emplear con mayor eficacia e interpretarlo en el contexto cada vez más extenso del software, entonces debemos plantearnos que los conocimientos de diseño existentes pueden ser empleados con eficacia para representarlos y codificarlos para que puedan ser accesibles a los demás.

La tarea de diseño mediante el uso de mecanismos existentes tales como los métodos de diseño orientados a objetos y patrones de diseño. Los patrones de diseño nos ofrecen la capacidad de abordar la experiencia con rapidez, sin duda es un proceso más rápido para escribir un nuevo patrón una vez identificado para el desarrollo de un nuevo método de diseño. Lo mismo sucede cuando se aplican a formas demasiadamente descriptivas, no hay descripción de las propiedades contenidas en base a los cuatro puntos de vista como son: función, comportamiento, modelado de datos y construcción.

Lo que puede quedar menos claro son los sistemas que se desarrollan mediante el uso de cualquier forma del diseño sistemático, ya sean los métodos o modelos. De hecho, los procesos de diseño de estos sistemas son probablemente muy informales y para sistemas más pequeños por lo menos, esto puede ser muy adecuado. Nos parece que los medios de hacer frente con el diseño de sistemas alrededor de la mayoría de las variaciones están en la forma de software, incluso esto puede requerir alguna adaptación en el tiempo.

Los conocimientos de diseño toman muchas formas y pueden comprender muchos "niveles" de abstracción, los problemas del conocimiento como la codificación no se limita al dominio de software, las características únicas de software se hacen presentes al diseñador con retos y oportunidades.

He empleado los conceptos de ocultamiento de información, el acoplamiento y la cohesión siguen siendo ampliamente utilizados en el diseño de software, a pesar de los avances posteriores bajo la forma del propio software, y en los procesos implicados en su desarrollo. También podríamos añadir la separación de las preocupaciones de esta lista, aunque se puede argumentar que esto tiene cierta superposición con el acoplamiento y la cohesión, ya que ambos se refieren a un sistema que se compone de módulos independientes.

Tal vez parte de su durabilidad, las ideas se encuentran en su propia independencia de las formas específicas de cualquier método o forma de ejecución. Los cuatro son conceptos relativamente abstractos, y todos siguen eludiendo cualquier medio simple de cuantificar.
En un nivel más detallado, los conceptos del patrón arquitectónico y diseño de patrón han proporcionado un medio útil de codificación de experiencias de implementación de estructuras en que trabajan, y también poseen características que permiten adaptarse y cambiar según sea necesario. Hay margen para ampliar el uso de patrones para hacer frente a las nuevas tecnologías, aunque estas últimas probablemente deben ser estables y constituidas antes que pueda ocurrir en cualquier escala, entre otras cosas porque el proceso de desarrollo es realmente un patrón que depende de la aparición de una comunidad de programadores con experiencia que son capaces de poner en común y compartirlas sus experiencias.

Los elementos principales en la codificación de la forma del conocimiento abarcan los siguientes aspectos:

- Estilo arquitectónico como marco de las actividades de diseño.
- Métodos de diseño que proporcionan una estructura en las prácticas procesales para el diseño de proceso propio.
- Patrones de diseño que proporcionan plantillas que describen las soluciones que pueden ser reutilizadas.

El estilo arquitectónico no influye directamente en la estructura de un proceso de diseño. Sin embargo, indirectamente, la elección del estilo, y las opciones que tal elección a continuación proporciona, significan que tiene una influencia importante sobre cómo el diseño es realizado. De hecho, podríamos argumentar que el desarrollo de ideas sobre la arquitectura de estilo arquitectónico forma una de las contribuciones más importantes en la evolución del pensamiento sobre el diseño de software, y sigue teniendo una influencia creciente sobre esto.

Los métodos de diseño, por el contrario, podrían ser un elemento de disminución del repertorio del diseñador. Muchos métodos incorporan supuestos sobre el ciclo de vida de desarrollo en cascada, sus formas de procedimiento no están bien adecuadas a la reutilización, su enfoque de procedimiento es apenas capaz de hacer frente a estilos arquitectónicos como la orientación a objetos.

Esto no quiere decir que los métodos de diseño ya no tienen un valor para el diseñador, pero es muy posible que su función ahora deba evolucionar, en caso de ser utilizadas para partes bien definidas de la tarea de diseño en general.

El patrón de diseño probablemente ofrece muchas más posibilidades de desarrollo y evolución, incluida la ampliación de su alcance para hacer frente a otras formas de la arquitectura orientada a objetos. Como ya se ha estudiado, la necesidad aquí es mejorar los procedimientos para su utilización. De manera

que, los patrones no parecen ofrecer la mejor manera de satisfacer las necesidades de los métodos, que pone énfasis en la reutilización.

Desde el punto de vista con miras hacia el futuro la tecnología siempre es un negocio arriesgado. El conocimiento actual y los mecanismos que usamos para transmitir a los demás, podrían beneficiar nuevas formas e ideas. Los problemas de diseño son problemas muy malos, y las características de software incluyendo su invisibilidad y complejidad, componen aún más los problemas a los que se enfrenta el diseñador del software.

El diseño del software seguirá siendo una de las formas más creativas e interesantes de la actividad de diseño, y la búsqueda de maneras de mejorar las técnicas, seguirá siendo una parte vital para el diseñador.

4.3.5.9 Lenguaje de diseño

Un lenguaje de diseño se puede seleccionar para un punto de vista de diseño sólo si es compatible con todos los elementos de diseño definido por ese punto de vista.

JSD utiliza un modelo del mundo real como base para la estructura del sistema, lo que significa que los cambios en el mundo exterior se pueden reflejar cambios en la estructura del modelo, y, finalmente, emergen como los cambios en la estructura del programa (s).

El marco más adecuado para describir el amplio proceso de diseño de JSD se describe en términos de las siguientes etapas:

- Una fase de modelado, en el que se analiza el problema y modelado en términos de las entidades constitutivas y las acciones que llevan a cabo, y donde estas entidades están representadas por procesos secuenciales de larga duración.
- Una etapa de red, en la que se desarrolla la estructura general del sistema del modelo añadiendo los detalles de las interacciones entre las entidades, y entre las entidades y el mundo exterior.
- Una fase de ejecución, en la que el diseño abstracto se proyecta sobre un diseño físico.

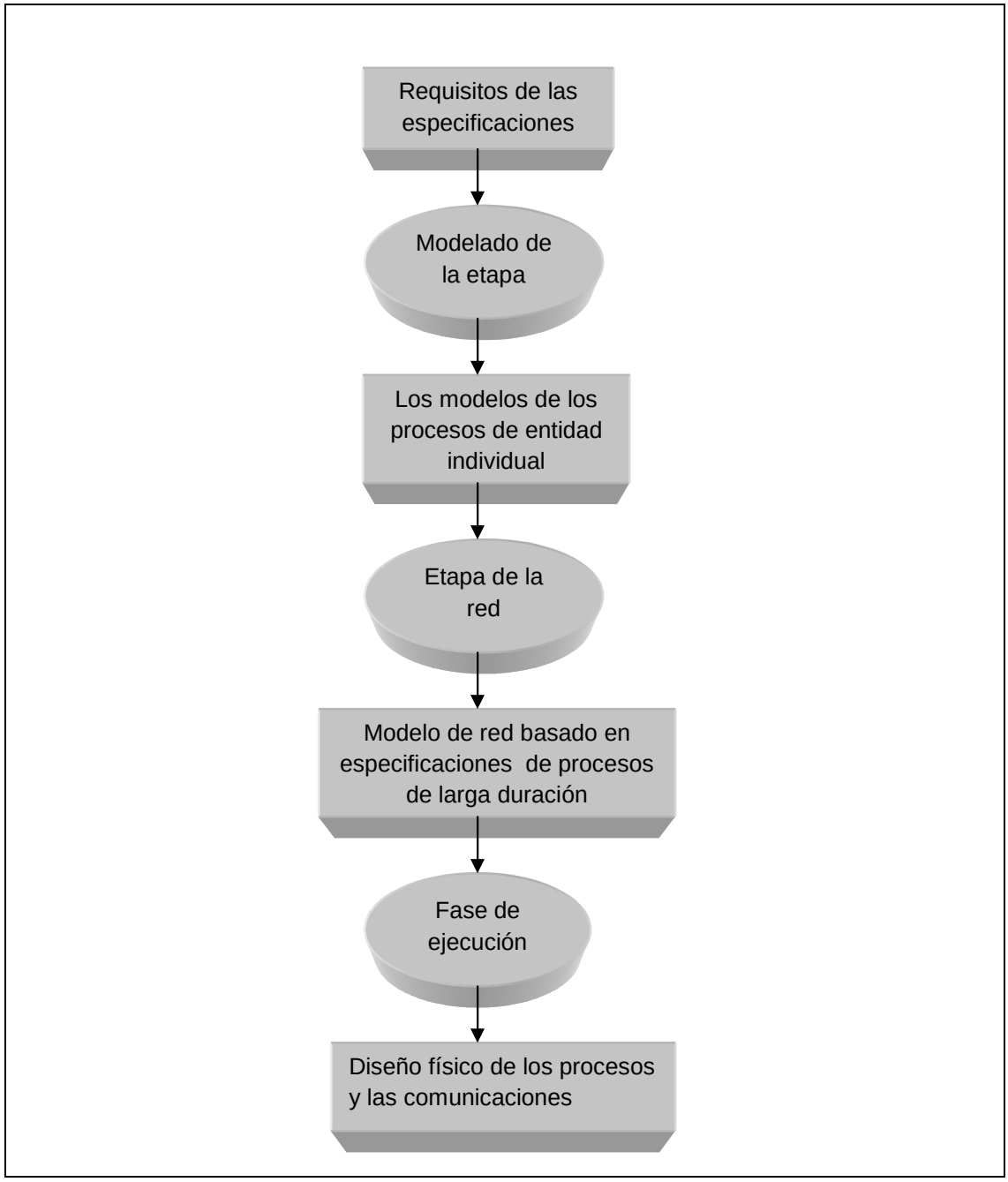


Figura 62.Modelo de transformación de alto nivel de JSD. Adaptado de [David Budgen]

JSD (Sistema de desarrollo de Jackson) se utiliza para describir la "evolución de una entidad a lo largo de un período de tiempo. En este contexto, la entidad es un elemento activo que se identifica a través de las operaciones del proceso de modelización en sí. Puede ser considerada como una forma de agente de transformación en el sistema.

La notación utilizada para un Diagrama de Entidad-Estructura (ESD) es la forma estándar utilizada para un diagrama de estructura, y la principal novedad en cuestión es simplemente hacer una interpretación diferente de los otros elementos del diagrama, a fin de adoptar un punto de vista bastante diferente del modelo del sistema. La ESD proporciona un medio de describir el comportamiento de los procesos secuenciales.

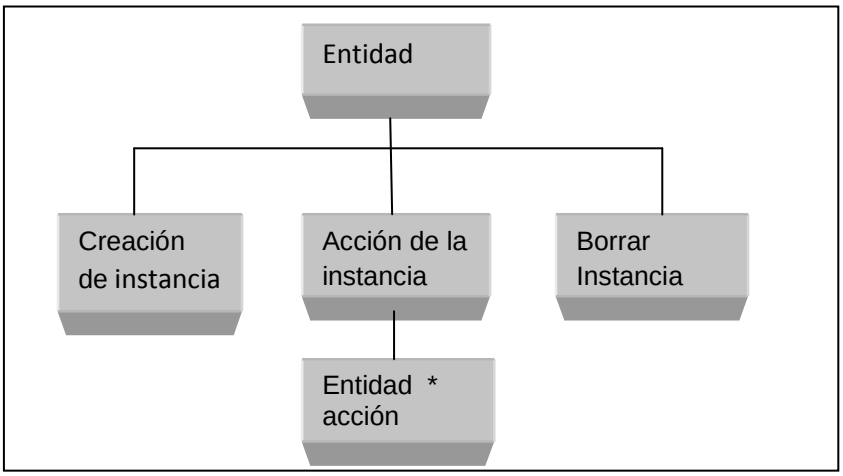


Figura 63.Diagrama de la estructura de la entidad. Adaptado de [David Budgen]

En la figura de la estructura de la entidad, la secuencia se refiere a:

- Creación de la entidad en el modelo.
- Acciones realizadas por la entidad durante su existencia.
- Eliminación de la entidad.

El diagrama de especificación de sistemas (SSD): Es un diagrama de red que identifica las iteraciones entre las entidades que conforman el modelo del sistema. Las iteraciones se producen por dos mecanismos entre procesos:

- Corriente de flujo de datos, en que los mensajes se transmiten de forma asíncrona entre las entidades interesadas. Una corriente de flujo actúa como una pila FIFO.
- Un vector de estado describe el estado local de un proceso en un momento dado. Mediante la inspección de ese vector de estado, una entidad puede obtener información requerida de una segunda entidad.

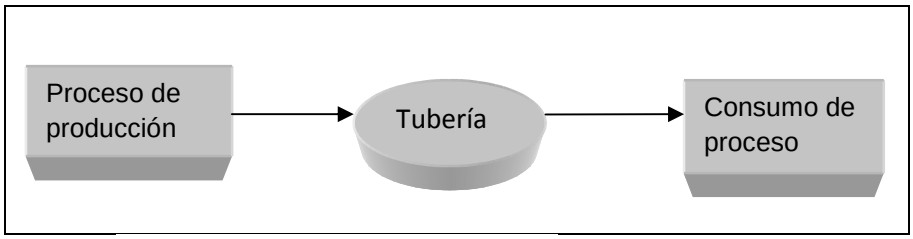


Figura 64.SSD, por una tubería (datos de flujo). Adaptado de [David Budgen]

El vector de estado consiste en atributos locales de un proceso incluyendo su contador de programa que se usa para indicar el estado actual de ejecución. El vector de estado se aplica en los procesos utilizados en el modelo del diseñador así como el proceso físico utilizado en la posterior ejecución.

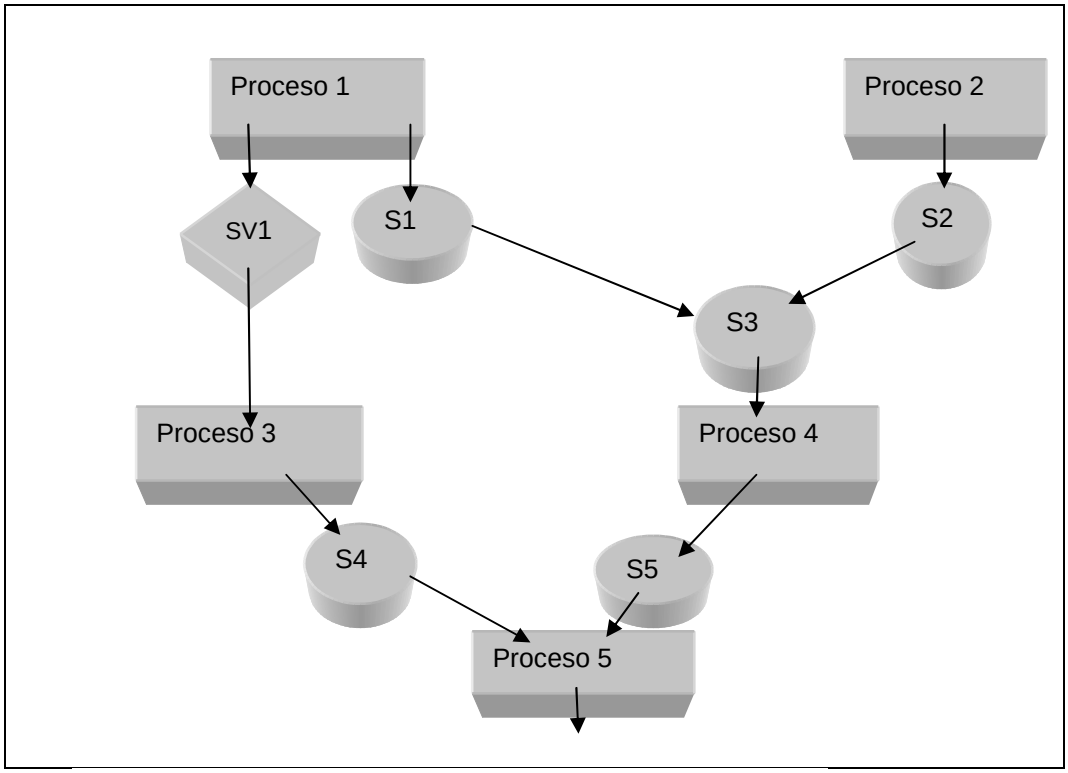


Figura 65.Segmento de diagrama de especificación de sistemas (SSD). Adaptado de [David Budgen]

La figura 65, describe un diagrama en un sistema. El círculo se utiliza para etiquetar un arco de flujo de datos y una entidad se representa mediante una caja rectangular, el rombo describe el vector de estado y se asocia a una entidad por medio de la etiqueta que se le asigne.

El proceso de JSD

Los procedimientos implicados en la parte del proceso de JSD han sido objeto de una cierta cantidad de revisiones.

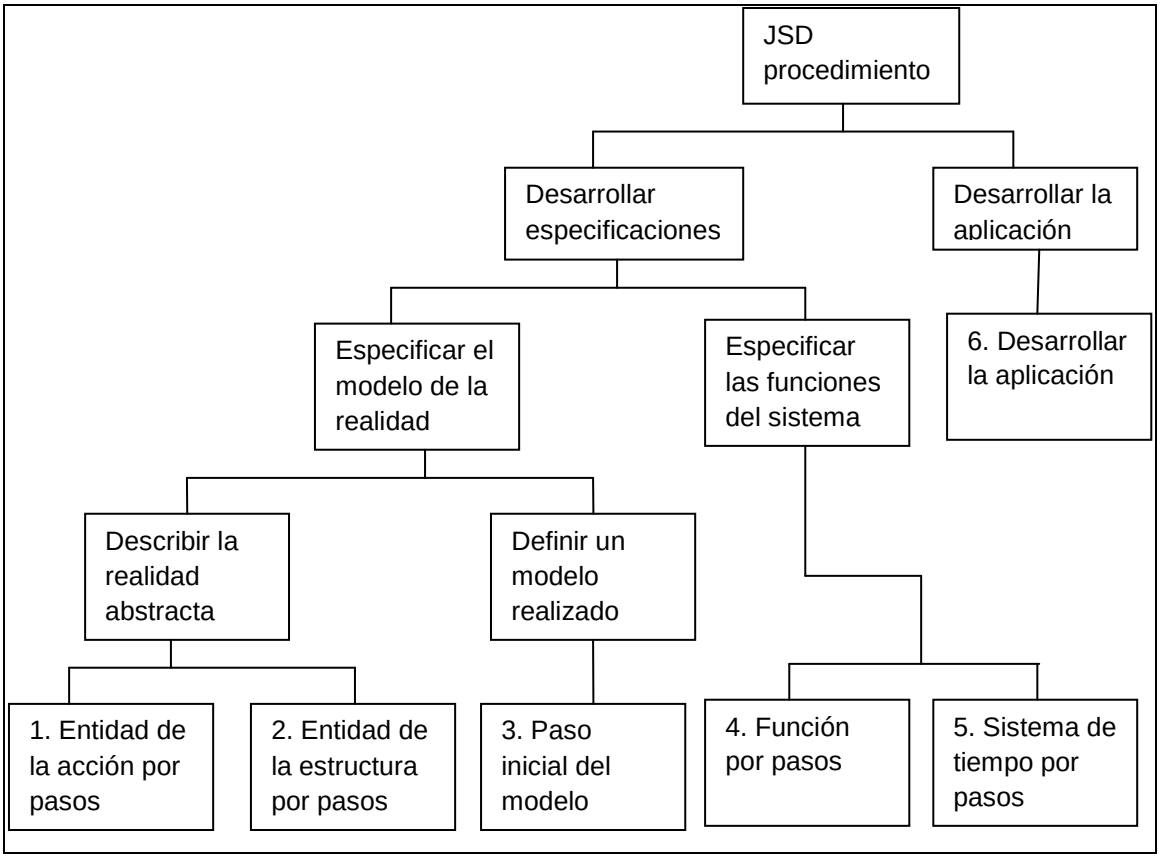


Figura 66.JSD procedimiento de descripción de Jackson. Adaptado de [David Budgen]

La etapa de modelado

Análisis de la entidad es decir, la entidad de acción y los pasos estructura de la entidad. Jackson recomienda que un diseñador deba comenzar esta tarea mediante el análisis de la especificación de requisitos con el fin de identificar a las entidades en el sistema, las acciones que realizan, los atributos de las acciones, y el tiempo-ordenamiento de las acciones.

Esta tarea de análisis de la entidad se basa en hacer un estudio del texto de los documentos de requisitos, aumentada si es necesario por las preguntas y las entrevistas con el cliente. Un objetivo principal para esta tarea es identificar las entidades del problema. Una estrategia consiste en analizar el texto de los documentos necesarios y cualquier otra descripción del problema que se puede obtener, en términos de los verbos y sustantivos.

Los verbos se pueden utilizar para identificar las acciones, mientras que los sustantivos son propensos a describir las entidades, aunque el proceso de extracción de las listas definitivas de estos requiere de mucho trabajo para refinar y referencia cruzada de los distintos candidatos. En el proceso de hacerlo, el diseñador también tendrá que eliminar este tipo de elementos extraños como sinónimos, "existencia" o verbos "estado", y entidades que no tengan relación directa.

JSD heurística

A continuación explicaremos las tres formas de heurística de JSD:

- Programa de Inversión: Es una técnica utilizada durante la fase de ejecución, cuando el diseñador comienza a considerar la organización detallada de una solución. Antes de eso, las actividades de diseño se basan en un modelo más abstracto y están menos preocupados con los detalles reales de la organización del programa físico.

- Vector de estado de separación: El método utilizado para manejar múltiples instancias de un proceso en el JSD es adoptar una versión más abstracta, y la tarea de diseño se refiere a la organización de los detalles de las estructuras de reentrada. El término que se utiliza para esto es la separación del estado de vectores. Dado que esta es una tarea que está preocupada con la asignación física de la solución, que se realiza normalmente durante la etapa final de ejecución.

- Backtracking: Ayuda a controlar los eventos inesperados que pueden ocurrir en la historia de la vida de la entidad, tales como su fin prematuro. Una vez más, el objetivo de esta heurística es la reestructuración de las pruebas de desordenamiento que están anidadas y de selección, y para manejar las iteraciones cuya realización puede ser incierto.

4.3.6 Conclusiones

- El concepto de un punto de vista de un modelo de diseño es una forma de capturar un conjunto determinado de propiedades de diseño, y cómo se proyecta a través del uso en una representación.

- Las clases principales desde el punto de vista de diseño, las formas de construcción, de comportamiento, funcional y modelado de datos.

- El uso de textos, diagramas y expresiones matemáticas son las tres formas básicas en las representaciones de construcción de diseño.

- Se han estudiado algunas de las principales formas del conocimiento del diseño de software y la experiencia codificada y transferida.

- El rol en el concepto de estilo arquitectónico es el suministro de un marco y vocabulario para las ideas de diseño de nivel superior, y para ayudar con la elección de la estrategia de diseño.

- Uso de métodos de diseño para las prácticas de diseño y las estrategias de codificación.

- La justificación del uso de soluciones de diseño.

- Se ha tenido en cuenta la naturaleza del proceso del diseño de software en términos de sus componentes y la posibilidad de proveer estos con la estructura.

- La justificación para utilizar un método de diseño también se ha considerado, y cómo el conocimiento del método puede ser un sustituto para el conocimiento del dominio, cuando sea necesario. Al observar la naturaleza del proceso de diseño de este modo, se ha tratado de determinar cuáles son las limitaciones en la forma y la aplicación de cualquier método del diseño de software que pretenden llegar a ser.

- Se han esbozado las principales técnicas que se utilizan actualmente en los métodos del diseño de software, y se han analizado algunas formas en que los atributos de un método de diseño pueden ser clasificadas y sus procesos pueden ser modelados.

- Todas las metodologías de desarrollo del software presentan modelos tendentes a producir software de la manera más eficiente y de la máxima calidad.

4.3.7 Puntos de vista de diseño

4.3.7.1 Introducción

Esta cláusula define varios puntos de vista de diseño para su uso en las SDDs Se ilustra la realización de estos puntos de vista de diseño en términos de selecciones de lenguaje de diseño, se refieren a diseño con puntos de vista, y establecen el lenguaje (notación y método) los nombres de estos puntos de vista neutral.

4.3.7.2 Punto de vista de contexto

El punto de vista contextual muestra los servicios prestados por un tema de diseño con referencia a un contexto explícito. Ese contexto se define por referencia a los actores que incluyen a los usuarios y otros interesados, que interactúan con el tema del diseño en su entorno. El punto de vista contextual proporciona un "recuadro negro" de la perspectiva sobre el tema de diseño.

Los diagramas de caso de uso sirven para mostrar funciones de un sistema software desde el punto de vista de sus iteraciones con el exterior, sin entrar tanto en la descripción detallada como la implementación de las funciones. Los casos de uso se utilizan tanto en la recogida como en la documentación de requisitos.

El diagrama de casos de uso se expresa en un nivel alto de abstracción. Los casos de uso se refieren a las interacciones que ocurren entre un sistema y su entorno y por lo tanto tienen un aspecto de comportamiento. Los casos de uso también se refieren a las tareas que el sistema realiza, lo que también se puede atribuir a una clasificación funcional. En el marco de punto de vista es un mecanismo de clasificación útil. Los casos de uso juegan un papel importante en el procedimiento de proceso unificado.

En el proceso unificado vamos a estudiar el flujo mediante las siguientes etapas:

- El flujo de trabajo de los requisitos

  El flujo de trabajo de los requisitos es una descripción técnica. Su objetivo es descubrir y averiguar lo que el sistema desea que realice a través de los casos de uso. Como definición de caso de uso podríamos decir que es una secuencia de acciones en la cual se incluyen variantes que puede realizar el sistema y como resultado ofrece a un determinado usuario que sea observable y tangible.

  En resumen los casos de uso especifican lo que debe hacer el sistema como y para que lo usen los determinados usuarios, la manera que los usuarios usan el sistema es mediante un caso de uso y unificando todos los casos de uso se tendrá la especificación del sistema.

  Una de las ventajas del caso de uso es que se pueden registrar tanto los requisitos funcionales: ¿Qué queremos que haga el sistema? Como los no funcionales: restricciones de tiempo, sistema operativo etc. Los casos de uso contemplan de una manera adicional la interfaz de usuario.

  Los casos de uso se refieren principalmente a la identificación de los límites de un caso de uso. La especificación detallada de un caso de uso es utilizada por los diseñadores que emplean una forma de plantilla basada en texto. Uno de los referentes para utilizar los casos de uso es proporcionar un mecanismo para la verificación de un modelado de diseño con los requisitos.

  Uno de los aspectos interesantes de emplear los casos de uso es que proporcionan un buen mecanismo para la verificación de un modelo de diseño con los requisitos. Ejecutar un diseño con un escenario de un caso de uso proporciona una vía a través del mecanismo que se forma mediante una relación directa

- Flujo de trabajo de análisis

  El flujo de trabajo de análisis del proceso unificado (UP) interpreta el análisis en el sentido convencional mediante cuadro negro, la producción de un modelo de lo que el sistema tiene que hacer, pero no cómo se hará.

Los objetivos del análisis de flujo de trabajo son los siguientes:

- Identificar las clases de análisis en los elementos del modelo en el dominio del problema.
- Producen casos de usos, el análisis de las clases interactúan para crear el comportamiento del sistema necesario para un caso de uso dado.

La identificación de las clases de análisis sigue siendo un problema difícil. La UP ofrece un marco útil para la estructuración de esto en la forma del caso de uso, ya que cada caso de uso puede ser separado a fin de identificar tanto las clases de análisis y la forma en que colaboran, por lo tanto proporciona una división de la tarea de análisis y otros medios de verificación.

La clase de análisis y la realización de casos de uso también se pueden agrupar en paquetes de análisis en los cuales se puede representar utilizando la notación de UML. Los paquetes contendrán clases, casos de uso u otros elementos de UML.

Los paquetes cumplirán los siguientes aspectos fundamentales:

- Coherencia, es decir, los elementos del mismo paquete estarán relacionados.
- Tendrán poca dependencia uno de otro, es decir existirán pocas conexiones entre elementos de paquetes diferentes.

Se establecen dos niveles de paquete:

1. Paquete de análisis

   Constituye la división del sistema del software desde el punto de vista del dominio. Cada paquete de análisis tiene varios subsistemas y puede servir en el trabajo de análisis de un equipo o de muchas personas. La descomposición en paquetes es útil cuando la carga de trabajo es muy elevada y su complejidad en los diagramas.

   Se pueden asignar paquetes a procesos del negocio o actores primarios y los casos de uso en cual se tenga relación de extensión, inclusión o generalización toda debe asignarse en el mismo paquete.

2. Paquete de servicio

   Son subdivisiones de los paquetes de análisis desde el punto de vista comercial, es decir con respecto a organizaciones (cliente). Los casos de uso son autónomos es decir que cada uno lo entrega al actor primario correspondiente. Los paquetes de servicio prestan las características siguientes:

   - Comprenden casos de uso.
   - El caso de uso tiene que ver con un actor.

- Son indivisibles, es decir, un cliente tiene un servicio de paquete o no lo posee.
- En un caso de uso pueden intervenir los paquetes que estén involucrados.
- Se pueden reutilizar entre distintas configuraciones del software.

La descomposición de los paquetes de análisis en paquetes de servicio se desarrolla utilizando paquetes de servicio a cada conjunto de caso de uso opcional.

4.3.7.2.1 Consideraciones de diseño

El propósito del punto de vista del contexto es identificar los servicios que ofrece un tema de diseño, sus actores (usuarios y otras partes interesadas que interactúan), para establecer los límites del sistema y delimitar el alcance con eficacia del tema de diseño de uso y funcionamiento.

Los casos de usos sirven para mostrar las funciones de un sistema del software desde el punto de vista de sus interacciones con el exterior y sin entrar en la descripción detallada ni en la implementación de estas funciones.

Los casos de uso se utilizarán tanto en la recogida y documentación de requisitos como el análisis.

Un escenario: Es una secuencia específica de acciones e interacciones entre los actores y el sistema objeto de estudio, se denomina instancia de caso de uso.

Un caso de uso es una colección de escenarios con éxito y fallo relacionados, que describe a los actores utilizando un sistema para satisfacer un objetivo. Los casos de uso son requisitos, ante todo son requisitos funcionales que indican qué hará el sistema.

En términos de tipos de requisitos, los casos de uso se refieren fundamentalmente a la funcionalidad o al comportamiento, pero también pueden utilizarse para otros tipos, especialmente cuando esos otros tipos están estrechamente relacionados con un caso de uso. También en métodos más modernos los casos de uso son el mecanismo principal que se recomienda para su descubrimiento y definición.

Los casos de uso definen un contrato de la manera en la que se comportará el sistema. Los casos de uso son documentos de textos, no diagramas, el modelado de caso de uso es, sobre todo, una acción de escribir texto, no dibujar. Sin embargo UML define un diagrama de casos de uso para ilustrar los nombres de casos de uso y actores y sus relaciones.

Los casos de uso de caja negra son la clase más común; no describe el funcionamiento interno del sistema, sus componentes o diseño, sino que describe el sistema en base a las responsabilidades que tiene. Los elementos

software tienen responsabilidades y colaboran con otros elementos que tienen responsabilidades.

Tipos de formalidad

Los casos de uso se escriben con formatos diferentes, dependiendo de la necesidad. Además del tipo de visibilidad de caja negra frente a caja blanca. Los casos de uso se describen con varios grados de formalidad:

- Formalidad breve: Se resume en un párrafo, normalmente del escenario principal.
- Formalidad informal: Formato de párrafo en un estilo informal. Múltiples párrafos que componen varios escenarios.
- Formalidad completa: La más elaborada. Se escriben con detalle todos los pasos y variaciones. Muestran más detalles y están estructurados; son útiles para entender en profundidad los objetivos, tareas y requisitos.

Nombre	Descripción
Nombre	Frase descriptiva
Actores	Actores que interaccionan con el sistema.
objetivos	Los actores tienen objetivos y utilizan las aplicaciones para ayudarles a satisfacerlos.
Precondiciones	Establece lo que siempre debe cumplirse antes de comenzar un escenario en el caso de uso. Las precondiciones no se prueban en el caso de uso, sino que son condiciones que se asumen que son verdad.
Postcondiciones	Establecen que debe cumplirse cuando el caso de uso se completa con éxito o bien el escenario principal o algún camino alternativo. Se debe satisfacer las necesidades de todo el personal involucrado.
Escenario Principal	El escenario recoge los pasos que pueden ser de tres tipos: - Una interacción entre actores. - Una validación, es decir, a cargo del sistema. - Un cambio de estado realizado por el sistema. Numeración de acciones principales en cada interacción del escenario principal.

Tabla 150.Especificación textual de los casos de uso

Los casos de uso se definen para satisfacer los objetivos de usuario de actores principales. El procedimiento básico es:

- Elegir los límites del sistema: Si no está clara la definición de los límites del sistema que se está diseñando, se puede aclarar definiendo lo que está afuera, los actores principales externos y actores de apoyo.
- Identificar los actores principales: Aquellos que tienen objetivos de usuario que se satisfacen mediante el uso de los servicios del sistema.
- Para cada uno, identificar sus objetivos de usuario.
- Definir los casos de uso que satisfagan los objetivos; nombrarlos de acuerdo con sus objetivos.

4.3.7.2.2 Elementos de diseño

Entidades de diseño: Elementos activos, actores externos que interactúan con la materia de diseño, incluidos los usuarios, otras partes interesadas y los sistemas externos, u otros artículos, servicios también llamados casos de uso.

Actores

La finalidad de un software es proporcionar información a personas, máquinas y dispositivos o software del exterior, cuyo conjunto denominaremos entidades exteriores. También hay entidades que piden funciones al software o le suministran información para que la traten.

Un actor es un conjunto de papeles de una entidad exterior en relación con el sistema de software. Por tanto, un actor no es la entidad exterior en sí, sino sólo los aspectos que tienen que ver con su interrelación con el sistema de software; un actor es la visión que el software tiene de una entidad exterior.

Desde este punto de vista, un actor es un conjunto de papeles ya que se considera que el actor desempeña un papel diferente en cada interacción de cada caso de uso en UML que tiene con el software. Además, si una entidad exterior forma dos conjuntos de papeles con poca relación entre sí, le corresponderán dos actores diferentes en este caso, el software no puede saber si los dos actores son o no la misma entidad exterior.

Una entidad exterior tiene que cumplir estas dos condiciones:

- Ser autónoma con respecto al software, es decir, que la actividad en que utiliza la información suministrada por el software no esté subordinada a la de éste.
- Mantener relación directa con el software o con entidades exteriores que desempeñan tareas subordinadas a la actividad del software.

Hay tres tipos de actores externos:

- Actor principal: Tiene objetivos de usuario.
- Actor de apoyo: Proporciona un servicio. Normalmente se trata de un sistema informático, pero podría ser una organización o una persona
- Actor pasivo: Está interesado en el compartimiento del caso de uso, pero no es actor principal ni actor de apoyo.

Los actores principales tienen objetivos de usuario que satisfacen mediante el uso de los servicios del sistema. Al contrario de los actores de apoyo proporcionan servicios al sistema que se está diseñando. No olvidemos que los actores principales pueden ser: sistemas informáticos, como proceso software etc.

Relaciones entre casos de uso

Entre los casos de uso se puede establecer tres tipos de relación:

- Relaciones de extensión: Se dice que el caso de uso A extiende el B si dentro de B se ejecuta A cuando se cumple una condición determinada. A tiene que ser un caso de uso que también se pueda ejecutar de forma separada de B, y debe tener el mismo actor primario que éste.

- Relaciones de inclusión: Un caso de uso A está incluido dentro de los casos de uso B, C, etc., si es una parte de proceso común a todos estos. A no es un caso de uso autónomo, en el sentido de que no tendrá actor primario, sino que siempre será puesto en funcionamiento por uno u otro de los casos de uso que lo incluyen. Su implementación no puede depender de estos, la inclusión de los casos de uso es esencialmente una forma de reutilización.

- Relaciones de generalización/explotación: Un caso de uso A es una especialización de otro caso de uso B, si A realiza todo el proceso de B, más algún proceso especifico.

Notación

Tanto para los casos de uso como para los actores, no se utiliza el símbolo de los clasificadores, sino símbolos especiales como los siguientes:

- Los casos de uso se representan mediante elipses de trazo continuo. A veces se agrupan todas las elipses dentro de un rectángulo que representa todo el software.

- Los actores se representan mediante una figura humana esquemática.

- Las relaciones de especialización entre actores y casos de uso se representan mediante el mismo tipo de flecha que en el caso de clases.

- Las relaciones de extensión y de inclusión entre casos de uso son estereotipos de la dependencia entre clasificadores, y se representan mediante las palabras clave extend e include, respectivamente.

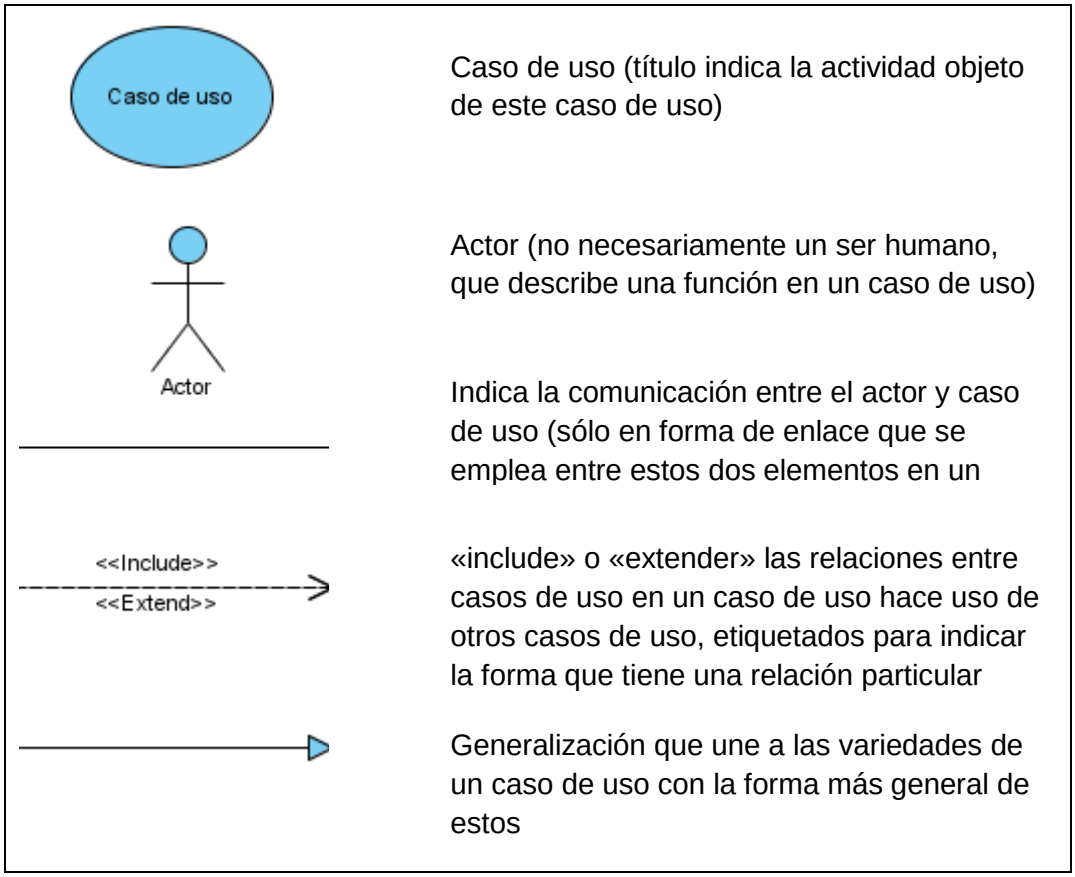


Figura 67.Elementos de diagrama de casos de usos UML. Adaptado de [David Budgen]

4.3.7.2.3 Ejemplo de lenguaje

Cualquier tipo de diagramas de recuadro negro se puede usar para lograr el punto de vista contextual.

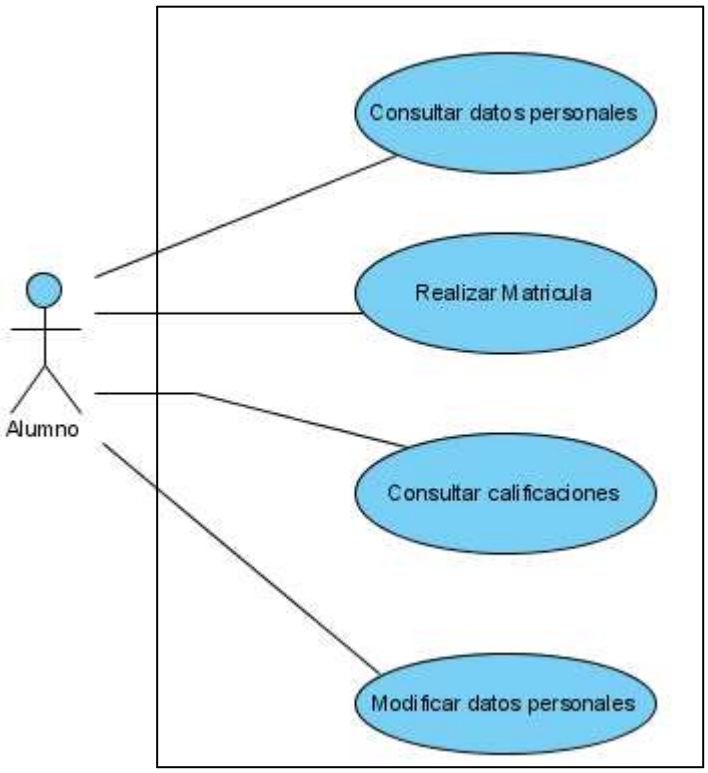


Figura 68.Diagrama de casos de uso

Nombre	Consultar datos personales
Actores	Alumno y personal Administrativo
Objetivo	Consultar o modificar los datos de un alumno
Precondiciones	El usuario ingresa a la aplicación como Alumno o Personal Administrativo
Postcondiciones	
Escenario básico	1. El usuario abre la sección como Alumno o Personal Administrativo. 2. El sistema solicita información de los datos. 3. Introducir datos del Alumno. 4. El sistema solicita confirmación de los datos introducidos. 5. El Alumno confirma los datos introducidos.

Tabla 151.Especificación de caso de uso

4.3.7.3 Punto de vista composición

El punto de vista de composición describe la forma en que el objeto es el diseño (recursivamente) estructurado en componentes y establece las funciones de las partes.

Definiremos un componente de software como: elemento del software que se ajusta a un modelo de componentes y puede ser desplegado de forma independiente y compuesto sin modificaciones de acuerdo a un estándar de composición. Este concepto de componente de software separa dos conceptos complementarios:

1. Modelo de componente que incorpora las interacciones específicas y las normas de composición.
2. La norma de composición se define, como los componentes que se pueden componer para crear una estructura más amplia.

Estos dos conceptos son importantes en términos de considerar cómo podemos diseñar los componentes y además como los componentes se deben diseñar para ser reutilizables. Se plantea de forma indirecta un problema sobre reutilización de componentes que dependen del estilo arquitectónico que se elija, y la medida en que la composición del proceso pueda necesitar una variedad de componentes de estilos arquitectónicos que se ven limitados a aquellos que cumplan con el estándar que se utiliza.

Los anteriores dos conceptos de componentes se diferencian en el nivel de abstracción para expresar las ideas y las características particulares de un componente.
En los últimos años ha surgido un enfoque de desarrollo de software denominado ingeniería del software basado en componentes (CBSE) que se basa en la reutilización.

Etapas intermedias en el proceso orientado a la reutilización:

Etapa	Descripción
Análisis de componentes	Con la especificación de requisitos se buscan componentes para su implementación.
Modificación de requisitos	Los requisitos se estudian usando información de los componentes que se hayan visto, por lo tanto estos componentes se modifican para reflejar componentes que se encuentran disponibles.
Reutilización del diseño del sistema	Los diseñadores tienen en cuenta la reutilización de los componentes y realizan un marco de trabajo para que se cumplan, si estos no están disponibles se tendrá que volver a rediseñar el sistema.
Desarrollo e integridad	Para la creación del sistema, el software que no se obtenga externamente habrá que crearlo.

Tabla 152.Etapas de CBSE

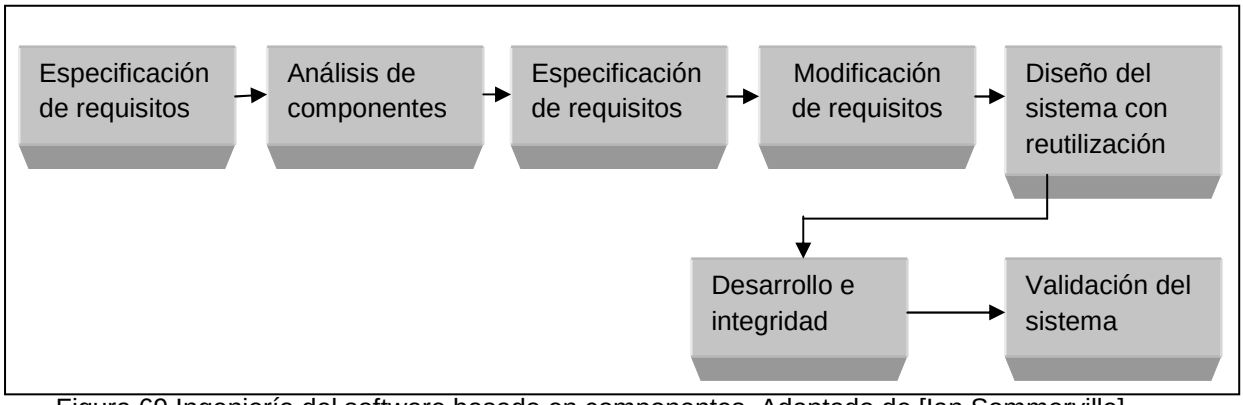


Figura 69.Ingeniería del software basado en componentes. Adaptado de [Ian Sommerville]

Sus principales subactividades dentro del proceso CBSE es:

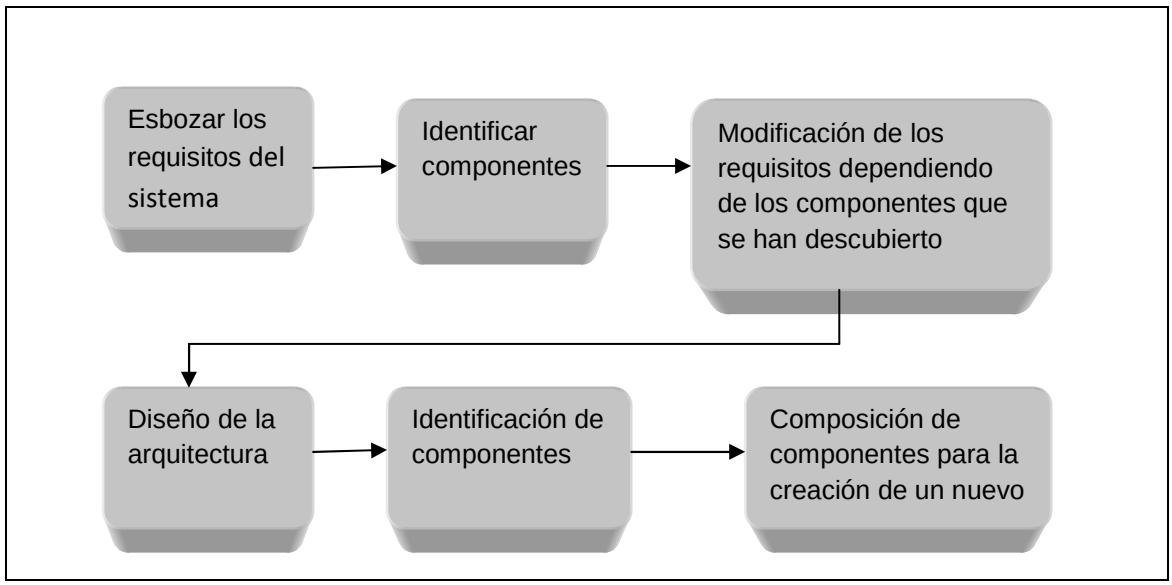


Figura 70.Proceso CBSE. Adaptado de [Ian Sommerville]

Las características de los componentes de cualquier forma requieren:

- Funcionalidad bien definida.
- Interfaces bien definidas.
- Dependencia.

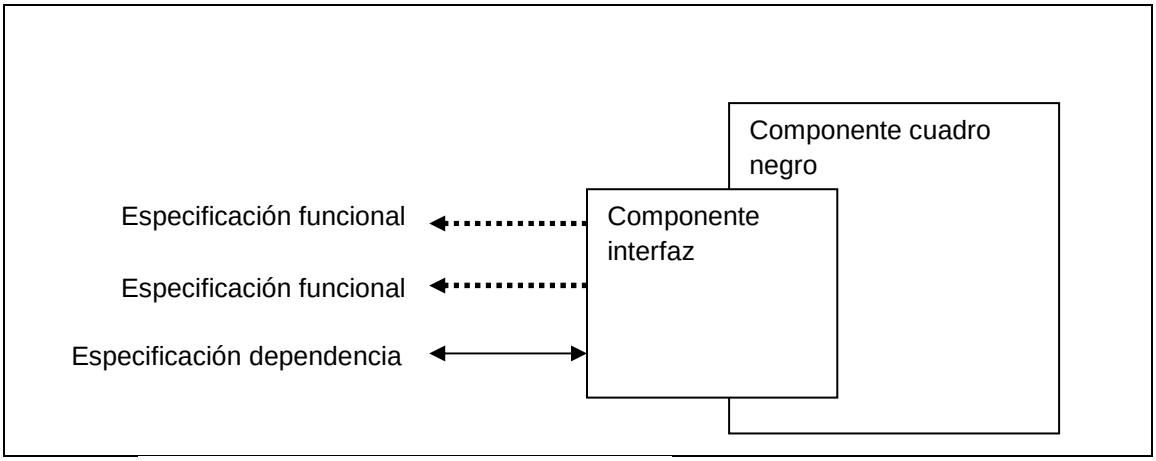


Figura 71.Características del componente del software. Adaptado de [David Budgen]

Debemos señalar aquí que el diagrama de componentes UML es una interpretación muy específica del concepto y de poco uso en CBSE. En efecto, tal vez porque el concepto carece de un componente de asociación con un estilo arquitectónico específico, en realidad no hay forma esquemática que sea ampliamente utilizado en el desarrollo de sistemas de CBSE.

Desde un punto de vista muchas otras formas de componentes tienen dependencias específicas para cada contexto. Por lo tanto la necesidad de asegurar que un componente de software puede ser tratado como un recuadro negro en la medida en que sea razonable y posible, con las dependencias como se hace en la especificación de interfaz, por lo tanto este es un elemento esencial.

Hay que tener en cuenta que si una estrategia CBSE se va a usar dentro de un contexto de negocio, especificaremos aspectos particulares en el contexto de negocio.

Contexto de negocio: Un enfoque que se basa en componentes para desarrollo del software es más que un simple conjunto de criterios tecnológicos, incluso más que un enfoque orientado a objetos, esto requiere distintas funciones de los diferentes actores involucrados y con el apoyo de algún tipo de mercado de componentes ya sean internos a una organización o externos.

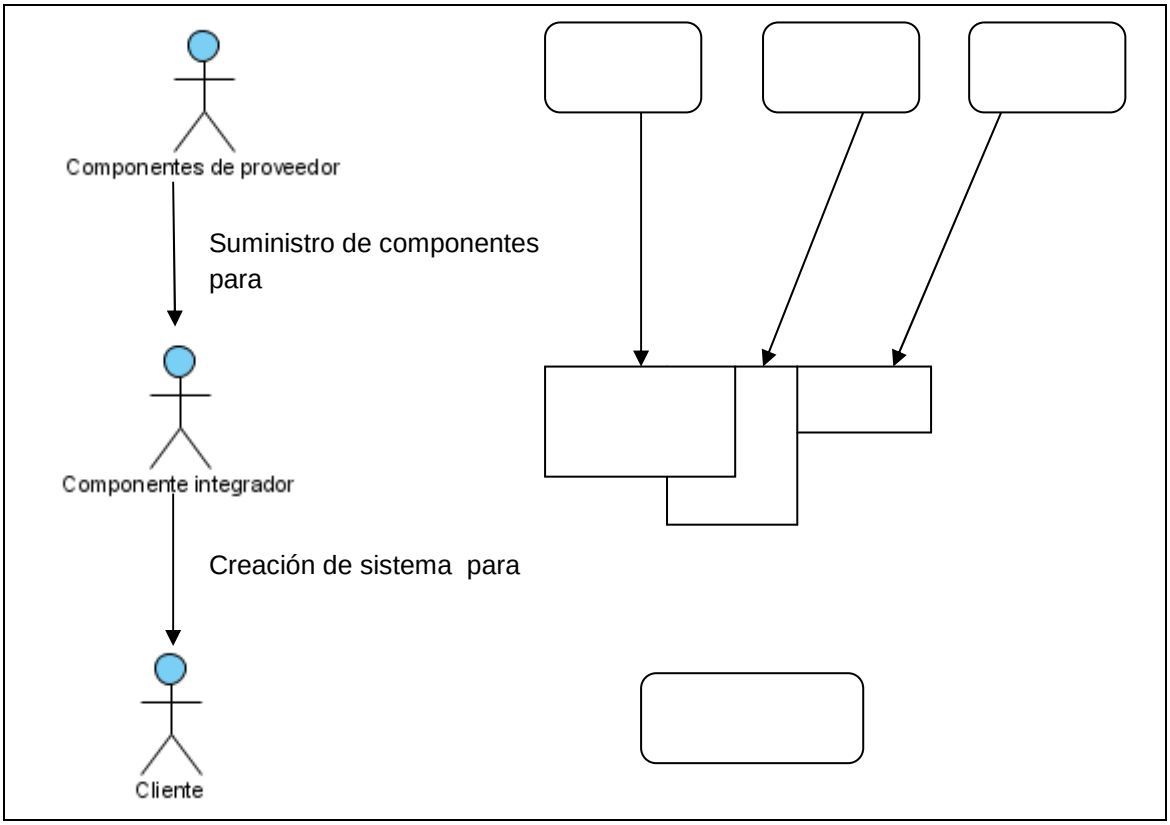


Figura 72.Los roles clave en el desarrollo de la CBSE. Adaptado de [David Budgen]

En la figura 72, podemos identificar tres tipos de rol, aunque cualquier persona u organización podrá realizar alguno de estos roles.

1. Proveedores de componentes. Los proveedores de componentes son necesarios para crear un mercado de componentes. Los proveedores de componentes deben ser capaces de indexar y describir sus productos, para satisfacer todas las normas, y para mantenerlos en el sentido de adaptarlos a las nuevas tecnologías y plataformas, así como la ampliación de su funcionalidad como sea necesario.

2. Clientes. El rol del cliente es identificar las necesidades y actuando como usuario final. Si bien en principio no debería haber ninguna razón por qué los clientes deben estar preocupados, o incluso conocer, si un producto está basado en componentes, hay algunas cuestiones que lo describen a continuación.

El cliente, para los sistemas basados en componentes debe tener en cuenta alguna serie de factores aunque algunos resulten relevantes para cualquier proceso de adquisición del software, pero es necesario abordar cuestiones a que CBSE se refiere, estas cuestiones incluyen: prestación de apoyo a largo plazo, existencia de una cadena de suministros, gestión de actualizaciones.

La prestación de apoyo se convierte en una cuestión muy compleja. En otros dominios la compra de componentes se emplea con frecuencia para proporcionar una mayor seguridad tanto para proveedor como el usuario final.

La cadena de una existencia de suministros, los proveedores del software utilizan herramientas básicas para la creación del software como compiladores. El problema que ofrece para el cliente es asignación de responsabilidad por cualquier problema que puede surgir en su uso, el rol integrador y la relación con el cliente es un elemento primordial en esto.

La gestión de actualizaciones es necesaria ya que cada proveedor puede actualizar los componentes de forma independiente, las consecuencias son para la gestión de mantenimiento general del sistema y actualización de programación. La naturaleza de este proceso puede hacer que sea difícil de manejar y, al mismo tiempo las responsabilidades técnicas pueden estar sobre todo con el rol integrador, el cliente tiene que ser consciente de este problema en la planificación de sus propias estrategias de prevención de riesgos y las actualizaciones del sistema.

Los diagramas de componentes desde el punto de vista de comportamiento tratan de describir vínculos entre los acontecimientos y las respuestas del sistema durante la ejecución; la conexión de un evento a una respuesta a cualquier condición necesaria. Esta forma de comportamiento tiende a ser muy abstracta por tanto son más propensas a ser afectadas con operaciones que en realidad se pueden propagar a través de una serie de elementos físicos de un sistema.

Los factores que influyen en el tiempo son muy desiguales y se resumen de la siguiente manera:

- Aspectos relacionados con la secuencia se describen con bastante facilidad.
- Descripciones de intervalo fijo son también bastante manejables, aunque su uso se limita principalmente a las características particulares de sistemas en tiempo real.
- Efectos de restricción son muy difíciles de capturar y describir el uso de las formas existentes.

El diagrama de componentes describe la composición física del sistema de software en componentes a efectos de construcción y funcionamiento. El término componente se utiliza para describir una unidad física de la aplicación con interfaces bien definida que se destina a ser utilizada como una pieza añadida de un sistema. Cada componente representa la aplicación de cierta clase de diseño del sistema.

Este componente tiene interfaces que se proporcionan para su uso por otros componentes, y las interfaces que se requieren para ser suministrados de otros componentes y ambos están representados por un pequeño grupo unido por una línea, con un identificador que representa la interfaz dada. Por lo tanto los componentes identifican objetos físicos que hay en tiempo de ejecución, de compilación o desarrollo, y tienen identidad propia y una interfaz bien definida de base para un componente.

Los detalles de las interfaces posibles, tienen que estar disponibles cuando un determinado componente se compila o cuando un conjunto de componentes están unidos entre sí, esta dependencia puede mostrar flechas discontinuas que pueden ser etiquetadas para mostrar los detalles de la dependencia en cuestión es decir en términos de unión, enlace. Un componente puede ser un objeto que se crea a partir de su clase padre o un componente puede ser una clase.

El componente UML es más que la creación de instancias directa de una clase u objeto ya que se puede representar mediante la agrupación de clases y objetos. Visto como un bloque de construcción puede considerarse como la realización de una caja blanca de estos conceptos. La caja blanca se logra a través de la información adicional proporcionada sobre las interfaces y las dependencias.

Los componentes pueden indicar:

- Tipo: se denomina componente de tipo y se dan en tiempo de compilación
- Instancia: se denomina componentes de instancia y existe en tiempo de ejecución.

Un componente se representa mediante tres rectángulos: identificador del componente (nombre del tipo), instancia (clases y objetos) y dos menores incrustados en el lado izquierdo del primero.

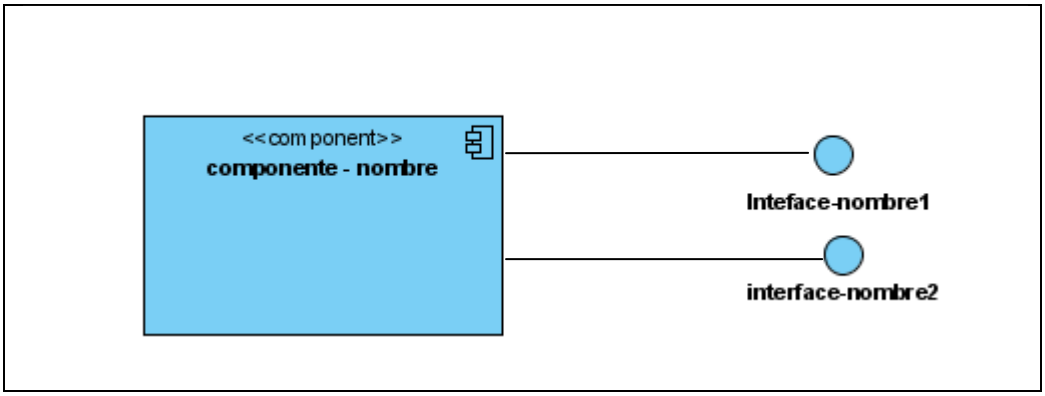


Figura 73.Diagrama de componente utilizado en UML. Adaptado de [David Budgen]

4.3.7.3.1 Consideraciones de diseño

Los desarrolladores de software y mantenedores utilizan este punto de vista para identificar los componentes principales del diseño, para localizar y asignar funciones, responsabilidades, o de otras funciones de diseño de estos componentes.

En la composición de un sistema de un conjunto de componentes el diseñador debe ser capaz de modelar y predecir su comportamiento agregado y funcionalidad, al realizarlo se encuentra con los siguientes problemas que se clasificaran de la siguiente manera:

- Funciones de solapamiento

  Esto ocurre cuando dos o más componentes son capaces de proporcionar una función del sistema en particular. La tarea del diseñador es determinar qué se debe realizar en la tarea, y asegurarse de que sólo es realizado por el componente seleccionado. Desde el punto de vista del diseño, no es deseable tener la misma función que proporciona diferentes elementos es decir: diferentes rangos y límites, y esto también crea un problema para el mantenimiento del sistema y mejora.

- Falta funcionalidad

  El problema surge cuando la funcionalidad total disponible del sistema es menor que la requerida. La solución a esto es encontrar otro componente o crear uno nuevo por tanto el problema es de identificación.

- Funcionalidad redundante

  Los componentes pueden proporcionar servicios más allá de los que son la base para elegir los componentes. La elección del diseñador es entonces incorporar esta funcionalidad, con posibles consecuencias adversas; esto se debe al solapamiento de funciones en un alto nivel de integración o encontrar de alguna manera la forma de aislar y deshabilitar las funciones deseadas, ya que esta puede ser la más sólida pero difícil de lograr.

- Desajuste arquitectónico

  Se presenta cuando existen desajustes entre las expectativas que cada componente tiene acerca de su contexto. En un nivel superior puede tomar diferentes formas en que los eventos se manejan. En un nivel inferior esto puede deberse a distintos lenguajes de programación utilizados para construir los componentes de uso en los diferentes pasos de parámetros en las llamadas a subprogramas. Los desajustes arquitectónicos pueden ser creados por las siguientes inconsistencias.

- Paquete de componentes

  Los componentes se pueden construir como componentes enlazables es decir: objetos, bibliotecas, etc. que necesitan ser integrados en el sistema ejecutable, o como componentes independientes, que pueden ser programados de forma independiente.

- Tipo de control

  Relación en que se organiza el control y se trasmite entre componentes, el cual se gestiona dentro de una aplicación o a través de un mecanismo orientado a eventos.

- Tipo de flujo de información

  Puede ser organizada como flujo de control de llamadas a métodos, como flujo de datos a través de la memoria compartida o mixta entre ambos flujos.

- Sincronización entre los componentes

  Si los componentes pueden bloquear la ejecución de otros componentes de modo síncrono o pueden seguir su ejecución sin tenerlo en cuenta de modo asíncrono.

- Tiempo de enlace

  Tiempo en que los componentes estén conectados a través de conectores durante la compilación, enlace, ejecución etc.

  Podemos sacar como conclusión mientras el desarrollo basado en componentes tiene el potencial para unir componentes en una gran variedad de fuentes y formas. El uso de componentes con un estilo arquitectónico puede ser la mejor opción de estrategia para adoptar, ya que se pueden presentar los 3 primeros problemas mencionados anteriormente, pero se pueden solventar. En cambio el 4 problema tenemos la falta de desajuste arquitectónico, desde estilos arquitectónicos pueden ser complementados de distintas manera aunque no garantiza que sea una buena opción.

El diseño de componentes para su reutilización también implica que el diseñador debe esforzarse por hacer un componente tan general como sea posible. Los criterios de acoplamiento y cohesión son ayudas útiles que se pueden utilizar para ayudar a pensar en lo que constituye una forma aceptable y el grado de generalidad en relación con la funcionalidad.

La reutilización de componentes software debe establecerse en un entorno que albergue los siguientes elementos:

- Base de datos de componentes: almacena componentes software, de igual manera información necesaria para recuperarlos.
- Sistema de gestión de bibliotecas: proporciona acceso a la base de datos.
- Sistema de recuperación de componentes software: permite a la aplicación cliente la recuperación de los componentes y del servidor biblioteca los servicios.
- Herramienta Case: integración de la reutilización de componentes en un diseño nuevo e implementación.

La reutilización del software se puede clasificar desde varios puntos de vista que nos permiten conocer variedad de facetas para medir y afianzar nuestros conocimientos acerca de ellos:

- Ámbito de desarrollo

  Desde el punto de vista de la reutilización se tiene que estudiar la procedencia del componente que se va a estudiar es decir si es interna o privada a la organización o aplicación, o externa o pública.

- Modificaciones

  La reutilización de elementos del software se utilizan sin modificar estamos haciendo hincapié en la reutilización de la caja negra ya que con la reutilización sistemática nos presta grandes beneficios. Se establece mayor beneficio la caja negra con respecto a la caja blanca. En el paradigma orientado a objetos se adquiere desarrollo con cierto tipo de modificación. El mecanismo del paradigma orientado a objetos como puede ser: herencia, encapsulamiento etc. Proporciona la capacidad de ser reutilizado con respecto a las modificaciones de comportamiento

- Ámbito de dominio

  La reutilización de elementos del software se utilizan sin modificar estamos haciendo hincapié en la reutilización de la caja negra ya que con la reutilización sistemática nos presta grandes beneficios. Se establece mayor beneficio la caja negra con respecto a la caja blanca. En el paradigma orientado a objetos se adquiere desarrollo con cierto tipo de modificación. El mecanismo del paradigma orientado a objetos como puede ser: herencia, encapsulamiento etc. proporciona la capacidad de ser reutilizado con respecto a las modificaciones de comportamiento.

- Nivel de gradualidad

  El elemento software reutilizable en el nivel de gradualidad hace hincapié si es un elemento atómico (grano fino) o si está compuesto por un conjunto de elementos reutilizables (grano grueso).

- Gestión
  Una organización gestiona la reutilización de forma sistemática si la planifica o incorpora como parte del proceso de desarrollo. La gestión es causal o ad hoc o si la reutilización se produce accidentalmente.

- Perspectiva cliente – servidor - sistema

  Perspectiva de desarrollo y reutilización (creación de aplicaciones de elementos que ya existen) o desarrollo para reutilización (procesos que crean elementos reutilizables). La realización en el desarrollo juega un papel crucial en el repositorio entendido como una base de datos que permite albergar la reutilización de los componentes. La orientación a objetos es una perspectiva cliente-servidor-sistema.

- Reutilización directa e indirecta:

  La reutilización de un componente si no se presentan intermediarios por el contrario es indirecta.

Ámbito de desarrollo	Modificación	Ámbito dominio	Nivel de granualidad	Entidad reutilizada	Perspectiva	Gestion	Reutilización
Interno (privado)	Caja blanca	vertical	fino	código	Sistemática	Cliente (con)	Directa
Externo (publico)	Caja negra	horizontal	grueso	Diseño	Casual (ad hoc)	Servidor (para)	Indirecta
				Test		Sistema	

Tabla 153.Tipos de reutilización del software adaptado [José Javier Dolado Cosín]

La función de la medición en la reutilización

Si la reutilización se realiza mediante la forma sistemática, para los gestores lo primero que se debería plantear es si tendrá que ver, costes y beneficios y en cuestión de las previsiones la organización aprobará o rechazará la reutilización a continuación se establecen los beneficios de la reutilización:

Nombre	Descripción
Disminución de los costes de desarrollo	Aumento de la productividad
Disminución de riesgos	Mejora de calidad
Menor tiempo de desarrollo	Fácil mantenimiento

Tabla 154.Beneficios de la reutilización

Los objetivos de métrica en reutilización:

- Realistas.
- Estimación de los beneficios de la reutilización.
- Proporcionan información a gestores y desarrolladores.
- Presentan valores de fácil entendimiento.
- Cumplimiento de ciertas propiedades como es la consistencia.

Los principales modelo de métricas de reutilización que se pueden utilizar:

Modelo	Descripción
Nivel de reutilización	Estimación de los que hemos reutilizado, diferenciando código reutilizado o código adaptado. En métrica interviene cuanto se ha reutilizado en la aplicación u organización se detalla en el modelo de coste/beneficio, para estimar los parámetros que intervienen.
Reusabilidad	Detalla el patrón de un componente reutilizable. Desde el punto de vista de repositorio al que se va a incorporar el elemento reutilizable. Se mejorarán aquellos atributos que están relacionados con la reusabilidad.
Influencia de la reutilización	Estima en cuanto se ha mejorado con la reutilización. En la métrica de influencia son muy importantes los índices de productividad ya que aportan beneficios cuando se compara el mismo evento hecho sin reutilización.
Económico	- **Modelo coste/beneficio:** Decide si se debe reutilizar o no. Se debe disponer de estimaciones coste total y coste de beneficio donde haya reutilización y donde no lo haya. - **Modelo ahorro de coste**: estima cuánto dinero vamos a ahorrar. - **Modelo de la recuperación de la inversión:** estima el beneficio neto de la reutilización tras haber consumido ciertos recursos.

Tabla 155.Modelos de métrica de reutilización

A continuación se presenta un enfoque de ciclo de vida abarcando desde las fases de definición y análisis de requisitos hasta la implementación. Para indicar los pasos a seguir en el diseño e identificar software reutilizable.

Atributo	Descripción
Portabilidad	El elemento del software no depende solamente de un sistema operativo o hardware
Fiabilidad	El elemento de software desarrolla la función de forma consistente y sin ningún error
Compleción funcional	El elemento de software satisface los requisitos presentes y los requisitos que se puedan dar en el futuro
Tratamiento de excepciones y errores	El elemento de software trata de aislar los errores
Ocultación de la información	El elemento de software es transparente al usuario en detalles de implementación

Fácil compresión	El elemento de software incluye documentación y comentarios en el código

Tabla 156.Atributos del software reutilizable

4.3.7.3.2 Elementos de diseño

Entidades de diseño: Los tipos de los componentes de un sistema: subsistemas, componentes, módulos, puertos y (siempre y es necesario) interfaces, las bibliotecas también, los marcos, los repositorios de software, catálogos y plantillas.

Relaciones de diseño: la composición, el uso y generalización.

Atributos de diseño: Para cada entidad de diseño, el punto de vista ofrece una referencia a una descripción detallada a través del atributo de identificación.

Entidades de diseño

- El componente es una entidad ejecutable independiente: el código fuente no se encuentra disponible. En el cual un componente no debe ser compilado antes de que sea utilizado con otros componentes del sistema.

- Los componentes se definen por interfaz: define los servicios proporcionados que el componente proporciona. Si estos no se encuentran disponibles el componente no funciona.

- Los componentes se realizan usando una aproximación orientada a objetos.

    1. Los componentes son entidades desplegables: no son compilados en un programa sino que se instalan directamente en una plataforma de ejecución.
    2. Componentes no definen tipos: un componente es una instancia no una plantilla que se usa para definir una instancia.
    3. Los componentes son opacos con respecto a las implementaciones: están definidos en su totalidad por la especificación de su interfaz. Para los usuarios de los componentes es invisible en la implementación.
    4. Los componentes son independientes al lenguaje: el lenguaje de programación orientado a objetos está definido por las clases de objetos y se relaciona con clases del mismo lenguaje, aunque también se puede realizar con lenguajes no orientados a objetos su implementación.
    5. Componentes estandarizados: se pueden implementar de cualquier forma excepto clases de objetos. Los componentes se deben ajustar a cualquier modelo de componente para su implementación.

La tarea de composición de componentes ensambla componentes cualificados, diseñados y adaptados para ser introducidos a una aplicación de una arquitectura establecida. Para lograr la composición de los componentes se debe tener presente lo siguiente:

- Modelo de intercambio de datos: Definen mecanismo que permiten iteración y transferencia de datos entre los usuarios y aplicación.
- Automatización: Implementación de herramientas entre componentes reutilizables que facilitan la interacción de macros y guiones.
- Almacenamiento de estructura: Heterogeneidad de datos deben ofrecer acceso con una estructura de datos y no como una colección de ficheros separados.
- Modelo de objeto subyacente: Distintos lenguajes de operación que residen en plataformas interoperables es decir que se pueden comunicar

Relaciones de diseño

En un diagrama de componentes se pueden observar las distintas relaciones que se pueden determinar entre componentes y otros componentes, objetos o procesos (objetos activos). En el caso de componentes no informáticos el significado de la relación es que un componente utiliza información de otro componente.

- En los componentes software se pueden distinguir dos tipos de relaciones.

- Relaciones en tiempo de desarrollo: Son asociaciones entre componentes que modelan dependencias las cuales se centran en tiempo de compilación o en tiempo de enlace.

- Relaciones de llamada: Son asociaciones entre componentes que sirven para modelar llamadas entre componente es decir que un componente (cliente utiliza servicios de otro proveedor) en tiempo de desarrollo la relación de llamada se desarrolla entre componente de tipo y su representación se realiza mediante diagrama de componente. En tiempo de ejecución se realizan entre dos componentes de instancia y se representan en los diagrama de despliegue.

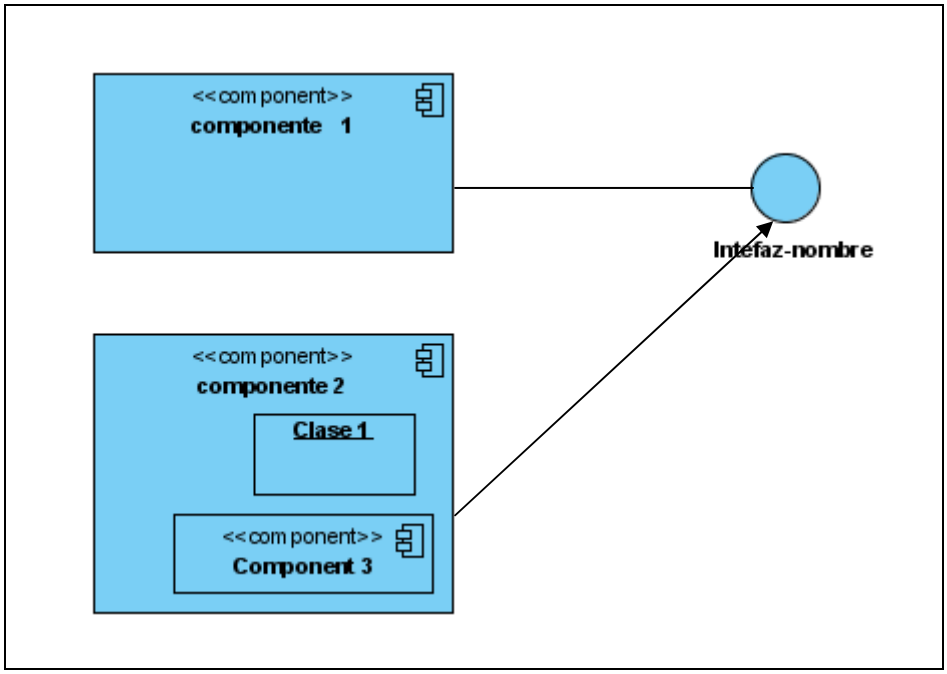


Figura 74.El componente 2 usa el componente 1 y contiene a su vez el componente 3 y un objeto de la clase 1. Adaptado de [David Budgen]

Diagrama de despliegue

Permite mostrar la arquitectura en tiempo de ejecución del sistema respecto a hardware y software. Este diagrama de despliegue se utiliza en el diseño y la implementación. Se pueden distinguir componentes como los de diagrama de componentes y nodos así como las relaciones entre ellos.

El diagrama de despliegue es más limitado que el diagrama de componentes es decir representa la estructura del sistema solo en tiempo de ejecución pero no en tiempo de compilación, pero resulta más extenso en el sentido que puede albergar más clases de elementos.

Los nodos

Representan objetos físicos existentes en tiempo de ejecución, sirven para modelar recursos que tienen memoria y capacidad de proceso, pueden ser: personas, ordenadores, dispositivos etc. Puede haber nodos tipo y nodos de instancia.

Relación en el diagrama de despliegue

Entre los nodos se establecen relaciones es decir existen comunicaciones entre ellos. Su representación es mediante líneas continuas en la cual realiza un estereotipo que indica el tipo de comunicación.

Atributos de diseño: Se definen para todos los componentes en una determinada área de dominio considerando los siguientes aspectos:

- No se limita el número de atributos que se pueden usar.
- En los atributos no se asigna ninguna prioridad.
- No se utiliza la función diccionario.

4.3.7.3.2.1Función atributo

Una declaración de lo que la entidad hace. Los estados del atributo, función de la transformación aplicada por la entidad para sus insumos para producir la salida.

Definen las propiedades de un objeto de datos y se utilizan en los siguientes aspectos:

- Nombrar una ocurrencia del objeto de datos.
- Describir la ocurrencia.
- Referencia a otra ocurrencia.

Los atributos definen uno o más atributos. Como un identificador este se convierte en una clave cuando se quiere encontrar ocurrencia de objeto de datos. El conjunto de atributos para un objeto de datos se establece mediante el contexto del problema.

4.3.7.3.3 Ejemplo idiomas

Diagramas UML de los componentes, el diagrama de HIPO. Composición en tiempo de ejecución también pueden utilizar los diagramas estructurados para cubrir este punto de vista.

Diagrama HIPO

Fue desarrollado por IBM como esquema de representación para un desarrollo jerárquico de arriba abajo y como una ayuda de documentación para productos comercializado.

Un diagrama Hipo contiene una tabla visual de contenido, un conjunto de diagramas generales y un conjunto de diagramas de detalle. La tabla visual de contenido es el directorio del contenido de diagramas en el paquete. Consta de un directorio con estructura de árbol, un resumen de los contenidos de cada diagrama general y una explicación de los símbolos utilizados.

Los diagramas generales especifican los procesos de un sistema en forma funcional, cada diagrama describe las entradas, los pasos de proceso y las salidas para la función en cuestión. Un diagrama general puede indicar la localización de los diagramas de detalle subordinados necesarios. Los diagramas de detalle tienen a su vez el mismo formato que un diagrama general.

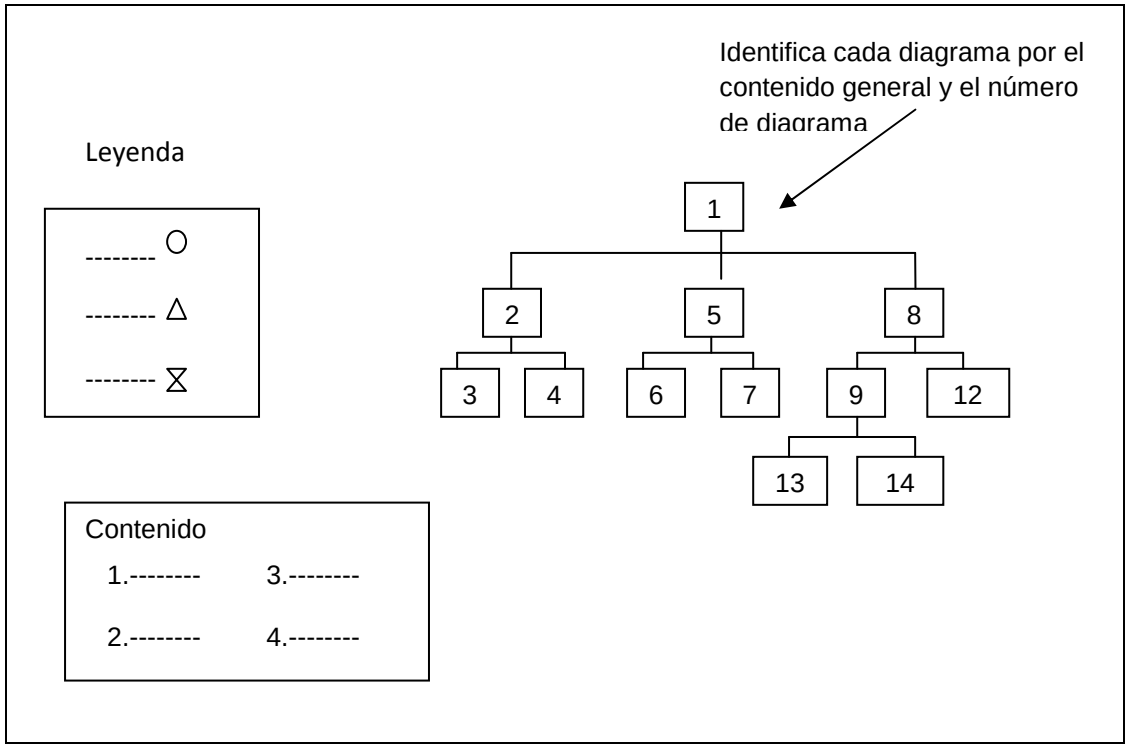


Figura 75.Diagrama Hipo. Adaptado de [David Budgen]

4.3.7.4 Punto de vista lógico

El propósito del punto de vista lógico es la elaboración de los tipos existentes y diseñados y sus implementaciones de las clases e interfaces con sus relaciones estáticas estructurales. Este punto de vista también utiliza ejemplos de instancias de tipos en esbozar las ideas de diseño.

Hacemos referencia desde el punto de vista de construcción ya que en el apartado siguiente, realizaremos el estudio desde el punto de vista lógico. Desde el punto de vista de la construcción se refiere a las relaciones que existen entre los elementos de la aplicación, aunque a un nivel muy alto de abstracción.

Las clases pueden estar relacionadas en un número de maneras. UML reconoce seis formas de relación (asociación, dependencia, flujo, generalización, realización y uso) con las distinciones adecuadas en la notación. Una descripción de los diagramas de clases UML, donde los roles son considerados principalmente como una caja negra pertenecientes a las actividades de modelización. Sin embargo, una de las características del paradigma orientado a objetos es que tales anotaciones se utilizan tanto para fines de análisis y también por un buen diseño detallado.

4.3.7.4.1 Consideraciones de diseño

El punto de vista lógico se utiliza para abordar el desarrollo y la reutilización de las abstracciones adecuadas y sus implementaciones.

Desde el punto de vista organizativo, los desarrolladores del software son los encargados de reunir los requisitos en el cual tendrán una formación y experiencia pero no conocen la actividad profesional de los usuarios, y como consecuencia de ello se denomina contexto del software que se describe de dos maneras:

- Modelo del dominio: Reúne las clases y tipo de objetos en el cual se estable lo siguiente:

  - Objeto de negocio.
  - Objeto del mundo real.
  - Acontecimientos.

  Para desarrollar el modelo de dominio, se usa el diagrama de clases UML. Y por ello al modelar el contexto, se trata de realizar un modelo de entorno del software y no del software, ya que este último se desarrolla dentro de otro proceso de componente como es el análisis y diseño.

- Modelo de negocio: Describe a grandes rasgos los procesos y entidades principales del software. El modelo de negocio describe las actividades de negocio en términos de caso de uso, entidades. Se usa diagrama de casos de uso y objetos, para explicar los casos de usos que se utilizan diagramas de iteración y actividades.

La diferencia entre el modelo de dominio y modelo de negocio es que no se puede sustentar que el modelo de dominio sea parte del modelo de negocio. Las clases del modelo de dominio se han conseguido mediante un estudio superficial de negocio, y el modelo de negocio describe los casos de uso después de que se hayan identificado las identidades que participan en el mismo, estos casos de uso se explican con mucho más detalle teniendo en cuenta aspectos organizativos más que informáticos.

4.3.7.4.2 Elementos de diseño

Entidades de diseño: Clase, interfaz, tipo de datos, objeto, atributo, método, clase de asociación, la plantilla y espacio de nombres.

Relaciones de diseño: Asociación, generalización, dependencia, realización, ejecución, instancia de composición, y la agregación.

El clasificador: Es la entidad básica del modelo estático. Un clasificador es más general que una clase, un clasificador es un conjunto cuyos elementos se

llaman instancias. El clasificador no tiene símbolo gráficos, sino que lo tienen sus estereotipos:

- Clase: El concepto de clase tiene el mismo concepto de programación orientada a objetos, las instancias son objetos y tienen identidad. Dos objetos que tienen el mismo valor que sus atributos son objetos distintos si se han creado como tales.

- Tipo de datos: Tipo base de cualquier lenguaje de programación, tiene operaciones asociadas y de igual manera que las clases, pero las instancias a diferencia de los objetos, no tienen identidad.

- Interfaz: Describe las operaciones de una clase que son visibles desde otras clases; por lo tanto dicha clase implementa la interfaz correspondiente.

Estereotipo

Un elemento en UML es una variante restrictiva de dicho elemento. Se pueden encontrar estereotipos que forman parte de UML y definidos que se refieren al diagrama, son un instrumento para ampliar UML; así se pierde portabilidad.

Un clasificador radica en que el estereotipo tiene algo en común, con lo que realiza la indicación una vez en el clasificador.

Paquete

Es una caja, contiene elementos, clasificadores, objetos, otros paquetes, entidades. En UML el concepto es diferente y mucho más extenso en el lenguaje java. Todas las aplicaciones deben contener al menos un paquete denominado raíz.

Cada paquete es visible o sea es reconocido de todos los demás paquetes, o solo desde algunos.

Tipos de relación entre paquetes

- Especialización: Un paquete A hereda de otro B todos los elementos de B, son casos más restrictivos de elementos de A.
- Inclusión: El paquete de A incluye el de B, entonces todos los elementos de B están también en A.
- Importación: Desde el paquete que importa se reconocen los nombres de los elementos del otro paquete visibles desde el exterior.

A continuación describiremos la clase y conceptos afines desde el punto de vista de la programación orientada a objetos. Una clase contiene un conjunto de objetos que a la vez poseen los mismos atributos y las mismas operaciones.

Los atributos y operaciones están vinculados a objetos individuales y de clase que no están relacionados con ningún objeto en particular de la clase. Los métodos y atributos pueden ser de instancia; el concepto de clase sobre programación orientada a objetos es el mismo concepto de clase en UML.

Diagrama de clases

Es un concepto básico de modelado de objetos y se centra en las relaciones que involucran a las clases. Un clase desempeña la función de plantilla que describe un concepto general y los objetos que son creados a partir de estos cuando se especifica instancia de una clase dada que posee un estado individual y la identidad.

Un diagrama de clases UML proporciona un medio de describir tanto las clases como las interacciones entre estas, particularmente basadas en la relación de los usos. En el diagrama de clases su elemento fundamental es la clase: es la descripción de los objetos del sistema que se pretende diseñar, objetos que comparten estructura, el comportamiento y sus relaciones. Las clases representan la información en el dominio del problema.

La estructura de los objetos viene especificada por los atributos mientras que el comportamiento lo especifica las operaciones del objeto de la clase. La notación de diagrama de clase UML es:

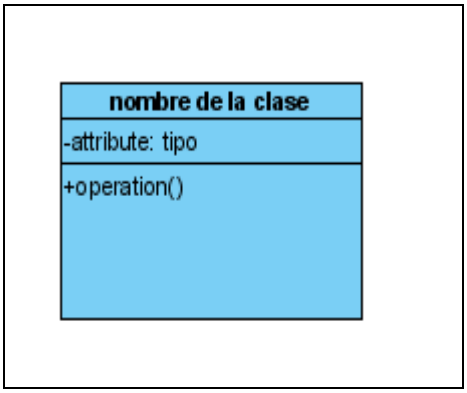


Figura 76.Notación de diagrama de clase UML

La clase se representa mediante unos rectángulos que se dividen en tres apartados:

1. Primer compartimiento: Nombre de la clase. Se puede indicar un estereotipo. Metaclase forma parte de UML y lo describiremos con más detalle más adelante. El nombre de la clase debe ser un sustantivo y en singular, que comience con una letra mayúscula.

   La posibilidad de reutilizar los nombre de las clases es muy importante ya que depende de ello, es decir si queremos reutilizar una clase de una librería que tenga muchos nombres la única referencia que tenemos es el nombre y por lo tanto si no se especifica un nombre adecuado será difícil de encontrar.

2. Segundo compartimiento: los atributos poseen un identificador y un tipo. Un tipo ya sea de instancia o clase, se define de la siguiente manera:

   Nombre ':' expresión tipo? '=' valor inicial

   Para los atributos se establece la visibilidad:

   - Pública: se representa anteponiendo el signo "+" esto significa si los atributos de los objetos de la clase son accedidos o invocados desde cualquier objeto del sistema.

   - Protegida: se representa anteponiendo el signo "#" al nombre del atributo, es decir que pertenezca a la relación generalización

   - Privada: se representa anteponiendo el signo "-" operaciones del propio objeto.

   - Paquete: se representa anteponiendo el signo "~" objetos del mismo paquete.

   Para los nombres de los atributos es necesario tener en cuenta:

   - Que comience por minúscula.
   - Si es atributo derivado, el nombre debe ir precedido de "|".
   - Cumplimiento de la reglas léxicas del lenguaje de programación, si no queremos que se cambie al llegar al diseño. Y sería igual para el tipo de dato como para el valor inicial.
   - Se pueden utilizar indicadores de multiplicidad como es el caso de las matrices dependiendo del lenguaje.

3. Tercer compartimiento: las operaciones, es decir n a los servicios de la clase.

   Se definen de la siguiente manera:

   Nombre '('lista de parámetros ')': tipo de retorno

   Indica que una operación es operación de una clase. El nombre de la operación, tipo de parámetros y retorno cumplan con las especificaciones del lenguaje de programación. Es indispensable que el nombre de las operaciones sea el más adecuado posible porque son la base del polimorfismo al igual que se aplica para el nombre de la clase.

   La sintaxis de la lista de parámetros es la siguiente:
   Tipo nombre ':' expresión de tipo '=' valor por omisión

   Donde entrada (in), si es de salida (out), si es de entrada /salida (inout), nombre es el nombre del parámetro formal, expresión de tipo depende del lenguaje, valor por omisión depende del lenguaje y es opcional.

Tipo de retorno: se usa cuando la operación devuelva un valor como resultado.

Consideramos distintos tipos especiales de clases pero no todos se pueden reflejar en UML.

1. Clases diferidas: son clases abstractas que poseen alguna operación abstracta o también denominadas clases virtuales.

2. Clases terminales: especialmente frameworks, nos interesa bloquear los cambios que se pueden realizar con la herencia, ya que al crear clases es posible que no se conozca la dependencia y restricciones que se heredan.

3. Metaclases: son clases cuyas instancias son clases. Raramente se encuentran implementadas en los lenguajes orientados a objetos. En UML. La metaclase es un estereotipo de la clase.

4. Clase plantilla: descriptor de clase formalmente igual a una clase, excepto algún término de su definición es un parámetro. Clase plantilla también se denomina clases genéricas en algunos lenguajes de programación. Aunque una clase plantilla no es propiamente una clase pero se pueden definir como subclase de una clase y por consiguiente la clase que se ha obtenido al dar valores a sus parámetros serán subclases de A.

5. Clase utilidad: en ciertos momentos nos encontramos con rutinas que no corresponden a ninguna clase de operación o datos que no corresponden a ningún objeto específico. Para incluir estas rutinas y datos en UML podemos definir una clase con el estereotipo utilidad e incluir como: datos, atributos, operaciones.

6. Interfaces: describe un conjunto de operaciones visibles de una clase sin indicar su implementación. Una interfaz no es una clase pero es una clase abstracta sin atributos y con todas las operaciones definidas. Cada interfaz específica solo la parte del comportamiento de una clase.

7. Las clases de entidades: Las clases constituyen bibliotecas de clases ya definidas y programadas donde encontramos clases que tengan relación con los objetivos del software. El disponer de bibliotecas de clases es una gran ventaja porque identificaremos clases y por consiguiente reutilización.

Clases por los expertos en el dominio

La importancia de la documentación revisada porque en ella conseguiremos entidades definidas explícitamente y en términos conocidos como pueden ser: objetos físicos, acontecimientos, departamentos, equipos, lugares etc. Por tanto se excluyen términos que son vagos o que corresponden a

entidades exteriores del sistema. Hay que tener en cuenta las clases ya implementadas para eliminarlas considerando las siguientes características.

- Clases sin atributos: nos indican que no hay información en el sistema.
- Clases sin operaciones: no tienen un papel activo ni pasivo en ninguna operación.
- Clases que poseen un atributo: puede ser que no sea una clase sino un atributo de otra.
- Clase con un solo objeto: es necesario ver si hay otras con atributos y operaciones similares y procurar unirlas. Es conveniente considerar los atributos y operaciones de este solo objeto como atributos y operaciones de una clase.

Los atributos de las clases de entidades son datos mencionados de forma explícita en los casos de uso, por consiguiente los usuarios lo usan directamente.

En el modelo de análisis es fundamental que sea independiente del lenguaje de programación. Los atributos son tipos abstractos y no soportados por un lenguaje específico, y a la vez establecer cuáles son esos tipos.

Los nombres de los atributos no deben presentar restricciones de nombre en ningún lenguaje de programación específico, esto es para evitar modificar los nombres al llegar al diseño, para ello establecemos un formato de nombres que sean compatibles con cualquier lenguaje de programación es decir atributos constituidos solo por letras y cifras sin que estas puedan figurar ni al principio ni al final.

Este formato vale para todos los nombres en general como pueden ser: clases, nombre de operaciones, parámetros etc.

Especificaremos casos especiales de atributos:

Atributo	Descripción
Atributos cuyos valores son compartidos por distintos objetos	Estos atributos serán atributos de clase.
Atributos no aplicables a Objetos de la clase	Se puede crear una superclase que tenga solo atributos que sean aplicables a todos los objetos y después una subclase que sea aplicable a todos los objetos respectivos.
Atributos con valor repetitivo	Es conveniente definirlos como una clase asociación de 1 a n, si son atributos compuestos o pares de atributos con la misma cardinalidad.
Atributos derivados	Se incluirán si figuran en salidas descritas en los casos de uso, siempre que sea un recurso para reducir la duración de los procesos a cambio de usar espacio de

	memoria o en la base de datos no se introducirán hasta el diseño es cuando se tiene en cuenta este punto de vista.

Tabla 157.Casos especiales de atributo

Relación asociación: Una asociación es una relación entre tipos es decir instancias de estos tipos indican alguna conexión. En UML las asociaciones se definen como la relación semántica entre dos o más clasificadores que implican conexiones entre sus instancias.

En un modelo de dominio en la inclusión se deben considerar los siguientes aspectos:

- Asociaciones, es necesario conocer el conocimiento de la relación.

- Asociaciones derivadas es decir las listas de asociaciones comunes.

- Inclusión de asociaciones, lista de asociaciones comunes.

  - A es una parte física de B.
  - A es una parte lógica de B.
  - A esta contenido físicamente en B.
  - A esta contenido lógicamente en B.
  - A es una descripción de B.
  - A es miembro de B.
  - A usa o gestiona a B.
  - A se comunica con B.
  - A es una propiedad de B.
  - A es una línea de una transacción de B.
  - A es un evento relacionado con B.

En un modelo del dominio con n clases del dominio diferentes, puede existir n(n-1) asociaciones entre diferentes clases conceptuales.

Representación de asociación en UML:

- Una asociación se representa como una línea entre clases con un nombre de asociación.

- Es inherentemente bidireccional, es decir que desde las instancias de cualquiera de las dos clases es posible el recorrido lógico hacia la otra, este recorrido es abstracto.

- Los extremos de la asociación pueden tener una expresión de multiplicidad es decir indican la relación numérica que existe entre las instancias de la clase.

- Flecha de dirección de lectura es opcional e indica la dirección de lectura del nombre de la asociación, es necesario especificar; no indica la dirección de la navegabilidad o navegación.

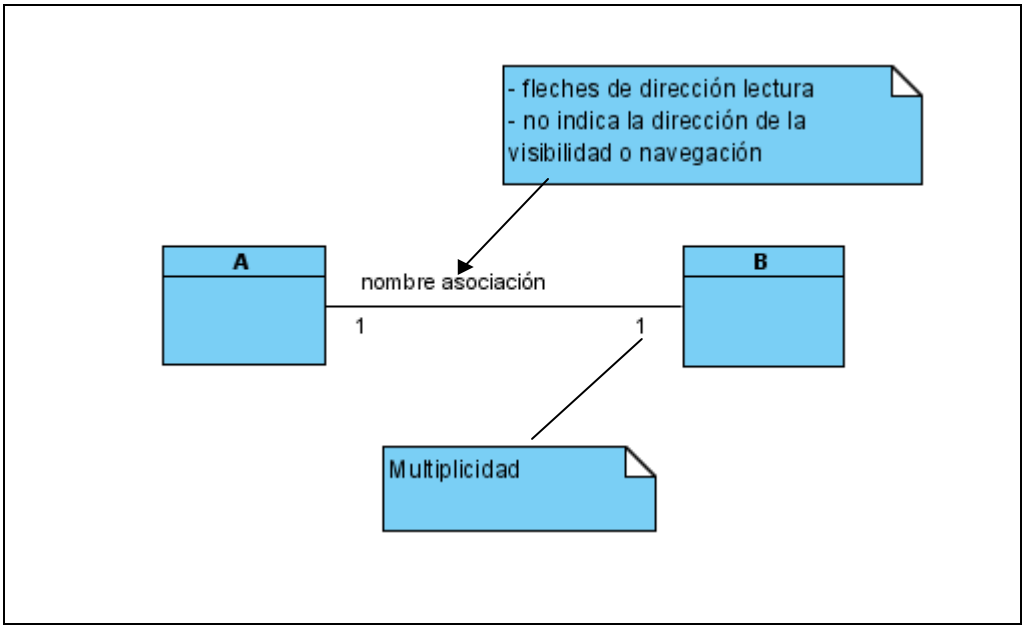


Figura 77.Notación UML para la relación de asociación

Partes que forman una asociación:

Nombre: debe comenzar con una letra mayúscula, ya que una asociación representa un clasificador de enlaces entre instancias.

1. Multiplicidad: Define cuantas instancias de una clase A pueden instanciarse en una clase B o viceversa. El valor de la multiplicidad depende de los modeladores y desarrolladores del software.Valores de multiplicidad.

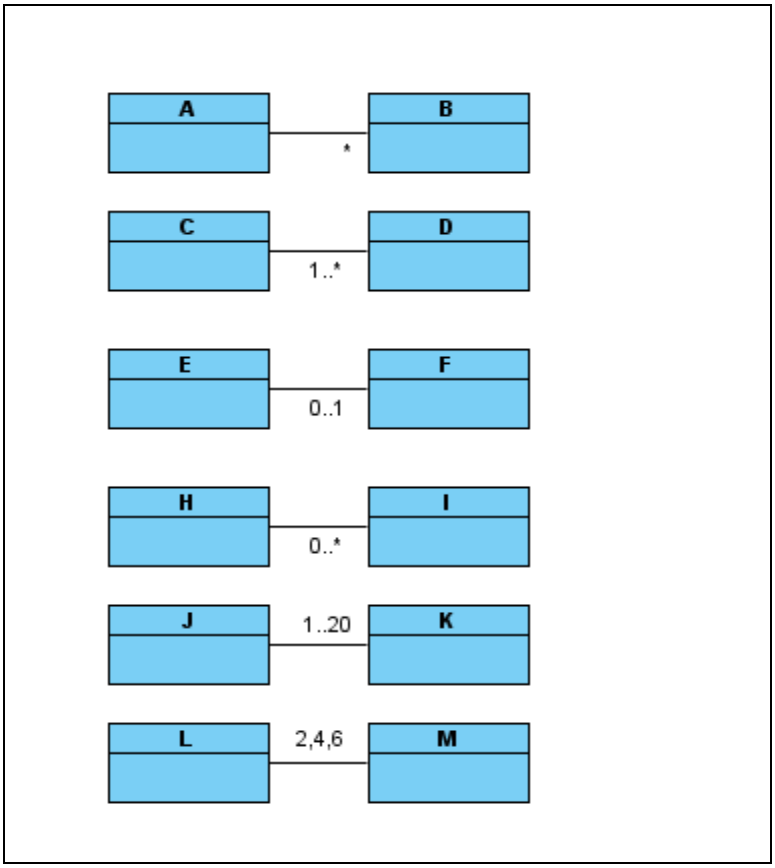


Figura 78.Valores de multiplicidad

Agregaciones: Una agregación representa una asociación asimétrica, en la que uno de los extremos juega un papel más importante que en el otro lado. Las agregaciones sólo se aplican a una de las funciones de una asociación, sin importar el número de clases que participan. Agregación se representa añadiendo un rumbo situado junto al agregado

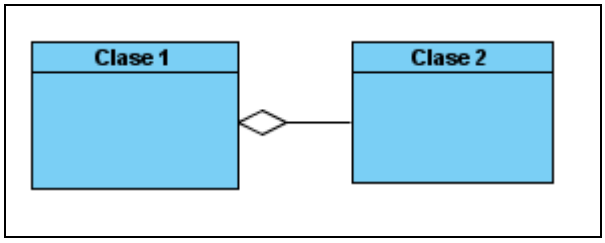


Figura 79.Relación agregación

Los siguientes criterios implican una agregación:

- Una clase es parte de otra clase.
- Los valores de los atributos de una clase para propagar los valores de los atributos de otra clase.
- Una acción en una clase implica una acción en otra clase.
- Los objetos de una clase son subordinados de los objetos de otra clase.

Como asociaciones, las agregaciones pueden ser varios valores que pueden estar en poder de las funciones de una agregación. Esto significa, en particular,

que la multiplicidad en la parte agregada tal vez mayor que 1. El concepto de agregación no asume una forma particular de aplicación.

Relación composición

Atributos son un caso particular de agregación de ejecución por valor son físicamente contenidos en el agregado. Este tipo de agregación se llama composición, y se representa en diagramas por un rombo negro.

Composición implica una restricción en la multiplicidad de la parte agregada: sólo puede tomar los valores 0 o 1. Un valor de 0 en el lado de los componentes que corresponden a un atributo sin inicializar.

La composición y los atributos son semánticamente equivalentes, y sus representaciones gráficas son intercambiables. La notación de la composición se utiliza en un diagrama de clase, mientras que un atributo participa en otras relaciones dentro del modelo. El siguiente diagrama ilustra la equivalencia entre la notación atributo y la notación de la composición:

"Una agregación de valor es semánticamente equivalente a un atributo, pero permite mostrar cómo un atributo participa en otras relaciones dentro del modelo".

Relación Dependencia

Indica la relación entre dos elementos de modelo y se representa mediante una flecha discontinua en el elemento.

Una relación de dependencia expresa, un elemento del modelo denominado cliente independiente para su implementación o funcionamiento de otro elemento denominado suministrador.

El símbolo de una relación de dependencia es una flecha de línea discontinua y punta abierta.

Existen diferentes estereotipos estándar, alguno de los cuales ya hemos visto y se puede definir otro más.

Derivan: Significa que un elemento se obtiene de otro por medio de un cálculo o algoritmo.

Amigo: Da acceso al cliente a los elementos de visibilidad privada contenido en el suministrador.

Refinar: Quiere decir que el cliente procede históricamente del suministrador, del cual es una versión nueva o enriquecida. Ejemplo una clase descrita en el análisis en el que se realizan cambios en el diseño.

Traza: Relaciona elementos que corresponden desde un punto de vista semántico al mismo concepto. Ejemplo un elemento y su implementación llamada, crear (create) y enviar (send).

Extender e incluir: Existe solo entre casos de uso.

Relación Generalización

Se deben tener en cuenta los siguientes aspectos:

- Diferentes clases de significado parecidos y tienen atributos y operaciones en común y se definirán como una superclase común a todas las clases que tengan atributos y operaciones en común.

  Los atributos comunes se deben colocar en la superclase y es necesario que se definan en todas las clases, las operaciones solo deben mantener el mismo nombre. La superclase es abstracta esto significa que sus operaciones deban serlo. Las operaciones que se desarrollen de la misma manera en todas las subclases estas serán operaciones de la superclase.

- Es importante situar los atributos y operaciones comunes lo más arriba de la jerarquía, aunque sus operaciones pueden ser abstractas.

- Si una subclase tiene todos los atributos y operaciones heredadas y esta subclase no tiene atributos propios la podemos eliminar, entonces la superclase será evidentemente concreta.

- No es conveniente que una superclase tenga una sola subclase. Por lo tanto en este caso es necesario eliminar la superclase.

En la generalización se representa mediante una flecha con una punta blanca hacia la superclase especificando tipo de generalización: completa, incompleta, solapada, disjuntas.

Restricciones de diseño: las restricciones de valor, las restricciones de relaciones de exclusividad, de navegación, juegos de generalización, la multiplicidad, la derivación, la mutabilidad, el valor inicial, clasificación, ordenación, estática, condición previa, posterior a la condición, y las limitaciones de generalización establecidas.

La restricción de multiplicidad depende del interés al usar el modelo. La multiplicidad denota una restricción de dominio si la relación se implementase, reflejase en los objetos o base de datos. El modelo de dominio no representa objetos del software, sino que la multiplicidad registra restricciones cuyo valor está relacionado con los intereses en la base de datos o construcción del software, es decir el dominio del mundo real.

4.3.7.4.3 Ejemplo idiomas

Diagramas de clases UML y diagramas UML objeto.

Diagrama de objetos UML

Su representación gráfica es muy parecida a las clases. Los valores de los atributos se indican mediante instancias y opcionalmente un nombre en el objeto va seguido de ':' y el nombre de la clase y subrayado. Se puede omitir el tipo de los atributos y el comportamiento de las operaciones, es decir los dos elementos mencionados anteriormente se conocen debido a la especificación de la clase.

Ejemplo

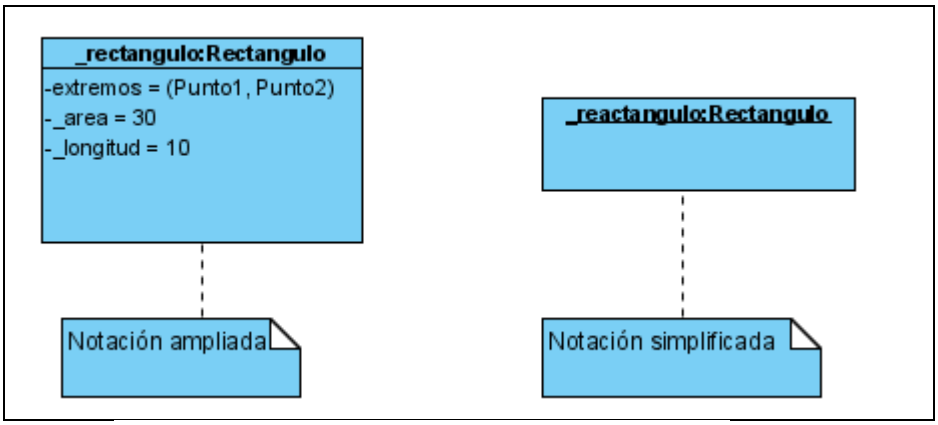


Figura 80.Objeto en notación ampliada y notación simplificada

4.3.7.5 Punto de vista de dependencia

El punto de vista de dependencia especifica las relaciones de interconexión y acceso entre las entidades. Estas relaciones son para compartir información, orden de ejecución, o la parametrización de las interfaces.

El diseño arquitectónico tiene como meta producir sistemas modulares de programación bien estructurados. A cada módulo de programación se le considera como una entidad definida que tiene las siguientes características:

- Los módulos contienen instrucciones, lógica de procesos y estructura de datos.
- Los módulos pueden ser compilados aparte y almacenados en una biblioteca.
- Los módulos pueden quedar incluidos dentro de un programa.
- Los segmentos de un módulo pueden ser utilizados por medio de invocar un nombre con algunos parámetros.
- Los módulos pueden utilizar a otros módulos.

En los módulos se incluyen los procedimientos, subrutinas y funciones, así como grupos funcionales de procedimientos, subrutinas y funciones relacionados. Los grupos de abstracciones de datos, grupos de programas y los procesos concurrentes.

La modularización permite al diseñador descomponer un sistema en unidades funcionales con el objetivo de obtener un ordenamiento jerárquico en el uso de las funciones, igualmente permite instrumentación de abstracciones de datos y el desarrollo independiente de subsistemas. La modularización puede ser usada para aislar dependencias funcionales, permite el desempeño de la programación para facilitar la depuración, pruebas, integración, ajuste y modificación de un sistema.

Existen muchos criterios que pueden ser empleados para definir la modularidad de un sistema, dependiendo del criterio usado puede resultar diferentes estructuras para un sistema dado.

- Criterio de modularización se puede incluir el criterio convencional es decir que cada módulo con sus correspondientes submódulos corresponden a un paso del proceso en la secuencia de ejecución.

- Criterio de ocultamiento de la información en que cada módulo oculta a otros módulos una decisión difícil o modificable del diseño.

- Criterio de abstracción de datos, en que cada módulo oculta los detalles de representación de una estructura de datos primordiales, debajo de las funciones que acceden y modifican dicha estructura.

- Criterio de acoplamiento y cohesión.

4.3.7.5.1 Consideraciones de diseño

El concepto de dependencia describe una relación entre dos componentes de tal manera que uno dependa del otro, pero esta relación no es totalmente visible.

Desde el punto de vista de diseño este es uno de los problemas: la captura de un tipo de dependencia es decir, un componente A invoca a otro componente B pero este no dice nada sobre otras dependencias que existan entre ellos, A y B comparten conocimientos acerca de ciertos tipos de datos y estructuras, al pensar en su solución el diseñador tiene que pensar en todas estas relaciones, esto puede llegar a convertirse en problema muy significativo cuando el diseño está siendo modificado durante el mantenimiento, el cambio de un diseño es muy difícil si el diseñador tiene solo una parte de los conocimientos necesarios para tratar la dependencia.

Para hacer un buen uso de una estructura modular, es necesario adoptar una práctica de diseño basado en la separación de preocupaciones.

Simplemente la definición de interfaces no es suficiente: Un diseñador realiza funciones dentro de los módulos de tal manera que su interdependencia se reduce al mínimo. Esta agrupación de resultados en una interfaz es menos compleja que el módulo: en el caso del hardware, puede ser simplemente que menos interconexiones son necesarias, y para el software se buscan otros criterios de los mencionados anteriormente para medir este factor.

Desde el punto de vista y también desde la perspectiva del software, encontrar un plan adecuado de la modularidad aplicado en la solución de un problema específico proporciona los siguientes beneficios:

- Los módulos son fáciles de reemplazar.
- Cada módulo capta un aspecto de un problema, por lo que ayuda a la comprensión y por tanto al mantenimiento.
- Un módulo bien estructurado puede ser reutilizado por otro equipo.

El uso exitoso de la modularidad relacionada con conceptos de calidad como: capacidad de pruebas, y si procede a la fiabilidad y usabilidad. Dos medidas de calidad útiles que han sido usadas para evaluar el grado de estructuración modular de software son el acoplamiento y la cohesión.

Se utilizaron para la complejidad de un sistema en términos de la forma e interdependencia de los módulos que lo componen. Tales formas modulares como paquetes, clase de java etc.

4.3.7.5.2 Elementos de diseño

Aunque los elementos de diseño los hemos profundizado en los dos anteriores apartados, por lo tanto no nos centraremos a fondo en su especificación.

Entidades de diseño

- Subsistema: Se ha establecido una descomposición del sistema del software en subsistemas y se han especificado las interfaces y las dependencias entre estas.

- Componente: Parte primordial, independiente y sustituible en un sistema cuya función satisface el contexto de una arquitectura definida.

- Módulo: Es un fragmento de un sistema software que se puede elaborar con relativa independencia de los demás. La elaboración de los distintos módulos la pueden desarrollar personas distintas y que todas ellas trabajen en paralelo. Tipos posibles de módulos:

1. Código fuente: Texto fuente escrito, que está contenido en un lenguaje de programación. Es el tipo de módulo como tal con mayor frecuencia y hace mayor hincapié en las diferentes técnicas de diseño. Su formato y organización dependen de la técnica y el lenguaje a utilizar.

2. Tabla de datos: Se usa para tabular ciertos datos de inicialización, experimental o de dificultad de obtención. La tabla de datos puede ser un módulo específico para cada forma de depósito y cuando se cambia el depósito solo es necesario cambiar el módulo.

3. Configuración: Un sistema se puede concebir en distintos entornos de trabajo según las necesidades de cada cliente. Interesa concentrar en un mismo módulo toda aquella información que permite configurar el entorno concreto de trabajo.

4. Otros: Un módulo en general puede servir para agrupar ciertos elementos del sistema relacionados entre sí y se puede tratar de forma separada del resto.

El formato concreto de cada tipo de módulo depende de la técnica y metodología o herramienta utilizada.

Relaciones de diseño

Haremos hincapié en el concepto de acoplamiento y cohesión

- Acoplamiento

  El acoplamiento entre dos módulos está influenciado por la complejidad de la interfaz, por el tipo de conexión y comunicación, se obtienen relaciones obvias a partir de una menor complejidad que de grandes y oscuras complejidades, es decir, las interfaces establecidas por bloque de control y datos, secciones comunes de entrada y salida etc.

  La modificación de un bloque común de datos o de control puede ser realizada en todas las rutinas que se encuentren acopladas a ese bloque. Si los módulos se comunican por medio de parámetros y las interfaces permanecen constantes esto implica que los detalles internos de los módulos necesitan ser modificados sin tener que modificar las rutinas que usan los módulos modificados.

  Las conexiones establecidas por medio de referencias a otros nombres de módulo se encuentran acopladas con las conexiones establecidas por medio de referencias a los elementos internos del módulo. La comunicación entre módulos incluye datos, elementos de control como son: interruptores, banderas etc. Así como modificaciones del código de un módulo hacia otro.

Forma	Característica	Conveniencia
Acoplamiento de contenido	Un módulo modifica los valores locales o las instrucciones de algún otro módulo. Ejemplo lenguaje de ensamblador.	Alto.
Acoplamiento de zonas compartidas	Los módulos son unidos en forma conjunta para estructura de datos por medio de zonas globales. Ejemplo lenguaje Fortran .	Moderada.
Acoplamiento de control	Incluye el pasaje de banderas de control ya sea como parámetros o en forma global o entre los módulos de tal forma que un módulo controla la secuencia de proceso de otro.	Alto.
Acoplamiento de datos	Incluye el utilizar lista de parámetros para pasar a los elementos entre rutinas.	Moderado.
Acoplamiento por zona de datos	Es parecido al de zonas compartidas con la diferencia de que los elementos globales son compartidos en forma selectiva entre distintas rutinas que requieren de los datos.	Moderado.

Tabla 158.Principales formas de acoplamiento

- Cohesión:

  La cohesión interna de un módulo se mide en términos de unión de los elementos dentro de un módulo.

Forma	Característica	Conveniencia
Cohesión coincidental	Ocurre cuando los elementos dentro de un módulo no tienen relación aparente entre cada uno de ellos	Baja.
Cohesión lógica	Los elementos de realizar operaciones que son lógicamente similares, pero que implican acciones muy diferentes internamente. Ejemplo bibliotecas de funciones matemáticas	Baja.
Cohesión temporal	Presentan muchas desventajas de los lógicamente unidos. Se encuentran en una conveniencia muy alta debido a que todos los elementos son ejecutados en un momento	Moderada.

	dado sin requerir ningún parámetro o lógica alguna para determinar que un elemento debe ejecutarse. Ejemplo inicialización de un sistema o programa.	
Cohesión en la comunicación	Los elementos de un módulo que contienen cohesión en la comunicación se refiere al mismo conjunto de datos de entrada y salida.	Moderada.
Cohesión secuencial	Cuando la salida de un elemento es la entrada para el siguiente. Una unión secuencial puede contener diversas funciones o partes de una función, debido a los procedimientos de los procesos en un programa pueden ser distintos del funcionamiento del mismo.	Muy alto.
Cohesión funcional	Representa un tipo fuerte y por ende deseable de unión de los elementos de un módulo debido a que todos los elementos se encuentran relacionados al desempeño de una función.	Alto.
Cohesión informacional	Ocurre cuando el módulo contiene una estructura de datos compleja. La cohesión informal es parecida a la cohesión de la comunicación ya que ambas se refieren a una sola entidad de datos, con la diferencia de que la comunicación implica que todo el código en el módulo es ejecutado en cada llamada al mismo, la cohesión informal requiere solo el segmento con cohesión funcional sea ejecutado al ser llamado el módulo.	Moderado.

Tabla 159.Principales formas de cohesión

Atributo de diseño

Estos atributos deben ser siempre para todas las entidades de diseño.

Nombre del componente

Dos componentes no pueden tener el mismo nombre. Al escoger el nombre tiene que reflejar su naturaleza. Esto simplifica la búsqueda e identificación de los componentes.

Tipo

Describe la clase de componente, muchas veces es suficiente con indicar el tipo de componente ya sea: subprogramas, módulos, procesos, datos, etc. Es necesario definir nuevos tipos basados en otros más elementales y además una lista coherente de tipos usados.

Propósito

Necesidad de que exista en cada componente. Para ello se hace referencia a un requisito que se trata de cubrir.

Función

Describe qué hace cada componente. Esto se puede detallar mediante la transformación de entrada / salida de igual manera se tratará si el componente es un dato.

Dependencias

Se enumeran los componentes que usan a este. Junto a cada dependencia se indica:

- Invocación de operación.
- Datos compartidos.
- Inicialización.
- Creación.

Recursos

Describe los componentes usados por ese componente que son externos a ese diseño tales como: impresoras, organización de la memoria, librerías matemáticas, posibilidad de interbloqueos etc.

4.3.7.5.2.1Dependencias de atributo

Una descripción de las relaciones de esta entidad con otras entidades. Las dependencias del atributo identifican los usos o requiere de la presencia de la relación de la entidad.

Independencia funcional

Como primer paso de la descomposición modular se puede establecer que cada función se podrá realizar en un módulo distinto, esto implica si el análisis está bien desarrollado y las funciones son independientes.

Al descomponer un sistema en módulos es necesario que exista relación entre ellos, se trata de reducir las relaciones entre módulos al mínimo. Una mayor independencia redunda en una mayor facilidad de mantenimiento o remplazar un módulo por otro y aumenta la reutilización del módulo.

- Acoplamiento: El grado de acoplamiento entre módulos es una medida de la interrelación que existe entre ellos: tipo de conexión y complejidad de la interfaz.
- Cohesión: Busca un acoplamiento débil entre módulos esto implica la necesidad de que cada módulo tenga coherencia. Cuando se descompone un sistema se debe buscar un objetivo específico para cada módulo.
- Comprensibilidad: El proceso de diseño e implementación de un sistema hace que los cambios en el sistema sean más frecuentes. Los cambios continúan durante la fase de mantenimiento hasta que se reemplace el sistema por otro nuevo. El primer factor que facilita la comprensión de un módulo es su independencia funcional. Considerando los siguientes factores:

    - Identificación: Elección adecuada del nombre del módulo y de los nombres de cada uno de los elementos, facilita la compresión.
    - Documentación: La documentación de cada módulo y del sistema en general debe servir para facilitar la comprensión especificando aspectos de diseño o implementación que por su naturaleza no pueden quedar reflejados de alguna manera.
    - Simplicidad: Soluciones sencillas son siempre las mejores.
    - Adaptabilidad: Al diseñar un sistema se pretende resolver un problema concreto. Un sistema que en mayor o menor medida se adapte a los nuevos requisitos que va imponiendo el cliente. La independencia funcional y comprensibilidad son dos cualidades primordiales en el diseño para posibilitar la adaptabilidad.

      En la adaptabilidad destacaremos los siguientes aspectos:

        1. Previsión: Resulta complicado prever la evolución futura que tendrá un determinado sistema.

2. Accesibilidad: Antes de abordar la nueva adaptación de un sistema es necesario conocer la suficiente profundidad.

3. Consistencia: Cuando se desarrollan adaptaciones sucesivas es vital mantener la consistencia entre todos los documentos de especificación, diseño e implementación para cada nueva adaptación.

4.3.7.5.3 Ejemplo idiomas

UML diagramas de componentes y diagramas UML como paquete que muestra las dependencias entre subsistemas.

Los componentes se realizan usando una aproximación orientada a objetos podemos tomar una pequeña orientación de cómo se usan:

El diseño orientado a objetos da como resultado un diseño que interconexiona objetos y operaciones no solo modularidad del procedimiento sino también la información exaltando el concepto de modularidad, abstracción y ocultación.

Los conceptos de objeto y operaciones de representación del mundo real son una mejora de abstracción de datos mediante la generalización de operaciones a tipos de datos que se introducen en lenguajes de simulación y construcción de prototipos.

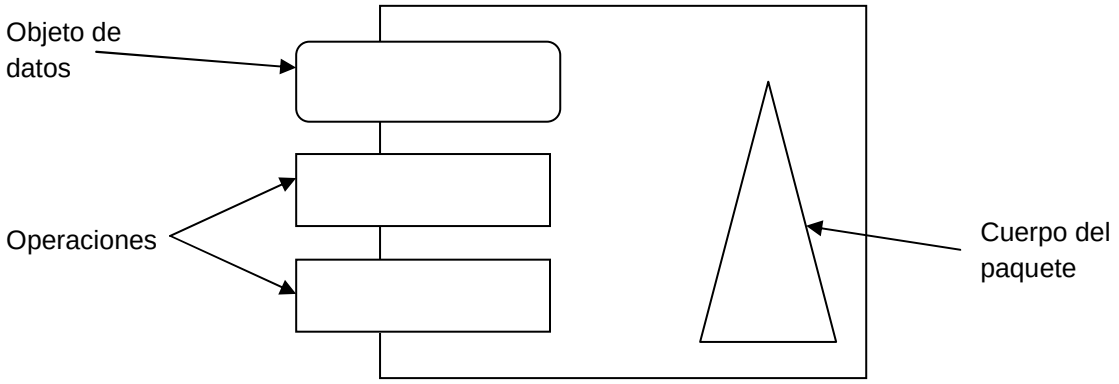


Figura 81. Diseño orientado a objetos

Para conseguir el diseño orientado a objetos se deben establecer:

- Representación estructura de datos.
- Especificación procesos.
- Establecimiento de los procedimientos de llamada.

La característica principal de un objeto es la unificación de los componentes de algoritmos y abstracción de datos, esta combinación de estos aspectos diferentes tiene que ser descrita mediante la utilización desde el punto de vista mediante estos aspectos como son: función, comportamiento, estructura,

modelado de datos. Se establecen interfaces para mostrar relaciones entre objetos y operaciones.

Los componentes de un programa (módulos) se definen y en el diseño orientado tiene que quedar identificado con un tipo de diagrama y además es necesario identificar las interfaces entre objetos y la estructura del programa. En el diseño orientado a objetos se representan los elementos en el contexto del lenguaje de programación o con ayudas gráficas.

La estrategia que se sigue es: el primer componente que hay que diseñar es el módulo de más alto nivel de donde está el origen de los procedimientos y estructura de datos, después se van conectado y representando cada uno de los componentes (objetos) con sus conexiones a través de interfaces.

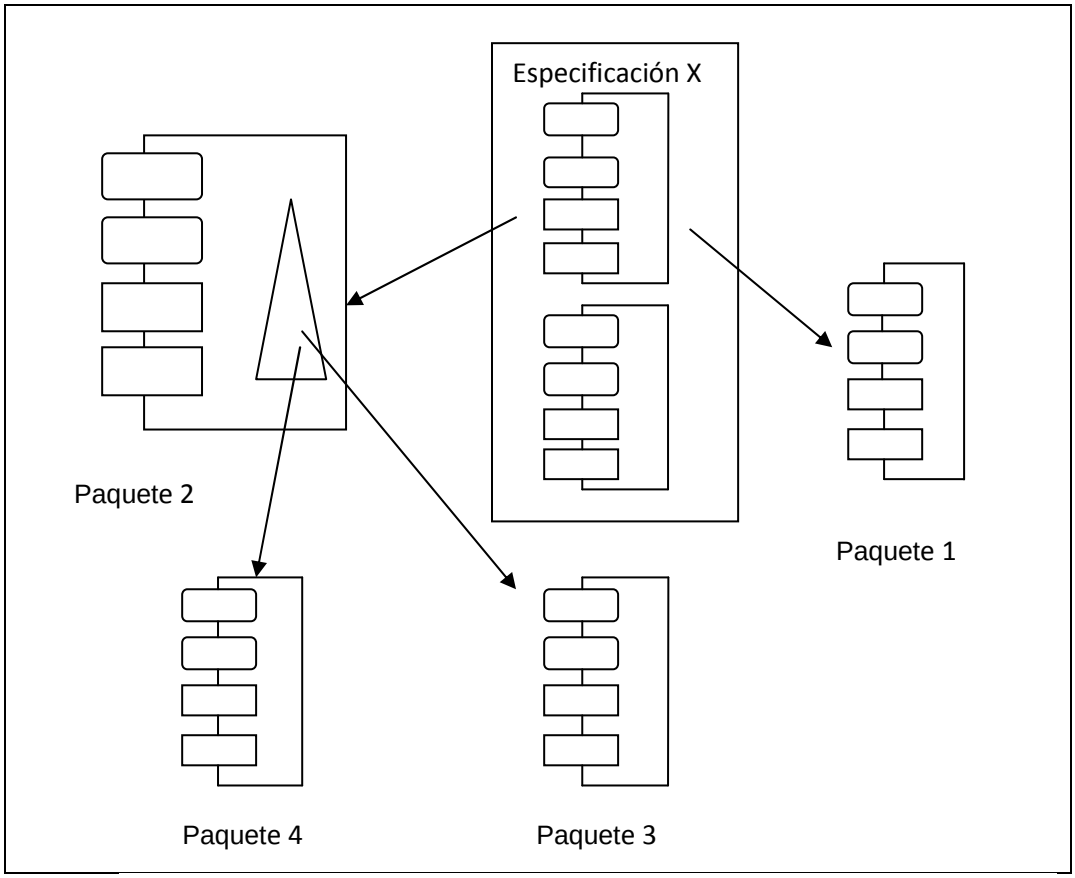


Figura 82.Componentes del programa (módulos) y conexiones de dependencia entre las interfaces de las componentes

4.3.7.6 Punto de vista de la información

El punto de vista de la información es aplicable cuando hay un contenido sustancial de datos persistentes con el tema de diseño.

La descripción de las estructuras de datos no tiene por qué ser un problema en términos de diseño arquitectónico, a menudo es un aspecto crítico de un diseño

detallado. Una vez más, no ayuda a un número de relaciones que pueden necesitar para ser capturado.

Estas incluyen las dependencias tales como: tipo (tanto en tipos de interés compuesto para crear otras nuevas, y en mecanismos como la herencia, que se utiliza para crear nuevas "clases"); secuencia (en términos de estructuras tales como árboles y listas).

Para algunas clases de problemas, la elección de estructuras de datos es muy importante y no es fácil separar de cuestiones funcionales. Debido a los puntos de vista que describen las estructuras de datos son esencialmente de naturaleza estática, y la manipulación por tanto, hay una serie de representaciones bien establecidas que son ampliamente utilizadas para el modelado de estas formas.

Modelado de estructuras de datos, que bien pueden ser impuestas por factores externos, se ve a menudo como más de una tarea de análisis de diseño, por esta misma razón. Sea cual sea el papel, las notaciones de modelado de datos tienden a ser utilizadas principalmente en papeles de la caja blanca, debido a la naturaleza relativamente detallada de la información involucrada.

4.3.7.6.1 Consideraciones de diseño

Preocupaciones principales son la estructura de datos persistente, contenido de datos, estrategias de gestión de datos, los datos de sistemas de acceso, y la definición de los metadatos.

Es la primera de las actividades de diseño y tiene una gran relevancia, ya que la estructura de datos repercute en la estructura del programa y en la complejidad procedimental y por tanto en la calidad del producto final.

Se seleccionan las representaciones lógicas de los objetos de la programación identificados en el análisis.

En los métodos de diseño se deben tener en cuenta los siguientes aspectos para el diseño de los datos:

1. Los métodos de análisis sistemático que se aplican al software deben aplicarse también a los datos.

    - Análisis de alternativas.
    - Revisión de solución.
    - Análisis y repercusión del impacto.

2. Identificar todas las estructuras y operaciones que se ejecuten sobre ellas.

3. Establecer y utilizar el diccionario de datos para determinar el diseño de datos.

4. Las decisiones de diseño de los datos a bajo nivel deben aplazarse a las últimas etapas del proceso de diseño.

5. En la estructura de datos, la representación se debe conocer solo por los módulos que hagan uso directo de los datos contenidos dentro de la estructura. La importancia de separación de la definición lógica de la implementación física, mediante la ocultación y desacoplamiento.

6. Desarrollo de la biblioteca de estructura de datos y de las operaciones que se puedan aplicar a ellos. Esto implicaría la reducción de trabajo de diseño usando recursos utilizables.

7. El diseño del software y el lenguaje de programación que se utilice debe mantener especificaciones y relación de tipos abstractos de datos.

Las bases de datos que se han convertido en los últimos años en un componente central de los sistemas de información por lo tanto es imprescindible contar con métricas para evaluar y controlar su calidad.

En la actualidad los temas relacionados con la calidad adquieren cada vez gran importancia en los ámbitos económicos y organizativos en especial en los sistemas de información por lo tanto se evaluarán distintos factores que influyen en la calidad de los mismos. Actualmente la medición del software es el medio más efectivo para comprender, monitorizar, predecir, mejorar su desarrollo y comprensión.

Métricas para bases de datos conceptuales

El modelo conceptual de datos es la base de todo trabajo de diseño posterior y el principal factor determinante de la calidad del diseño del sistema en general. Esto implica la importancia que tiene contar con métricas que permitan evaluar y controlar la calidad de los modelos conceptuales de datos.

El objetivo primordial es demostrar diferentes propuestas de métricas para evaluar los factores más influyentes en la calidad de la base de datos desde el punto de vista conceptual como desde el punto de vista lógico.

- Métricas de Moody

  Desde el punto de vista práctico. Se pretende ayudar a los diseñadores a elegir entre distintas alternativas y acomodar las distintas visiones implicadas en el proceso de modelado de datos.

Factor de calidad	Métricas
Compleción	- Número de requisitos de usuarios que no están representados en el modelado de datos. - Número de requisitos de usuarios que están definidos de una manera inapropiada. - Número de inconsistencias con el modelo de procesos
Integridad	- Reglas de negocio que no se cumplen en el modelado de datos. - Número de restricciones de integridad incluidas en el modelado de datos que no corresponden a políticas del negocio.
Flexibilidad	- Elementos en el modelo que están sujetos a cambios futuros. - Coste estimado de los cambios. - Importancia estratégica de los cambios.
Facilidad de comprensión	- Valoración de los usuarios sobre la facilidad de comprensión de los datos - Capacidad de interpretación en el modelado de datos por parte de los usuarios. - Comprensión en el modelado de datos para los desarrolladores de aplicaciones.
Corrección	- Número de instancias de redundancias en el modelado de datos.
Simplicidad	- Número de entidades. - Número de entidades e interrelaciones. - Número de constructores.
Integración	- Numero de conflictos en el modelado de datos corporativo. - Número de conflictos con los sistemas existentes. - Valoración de los representantes de todas las áreas de negocio.
Implementabilidad	- Valoración del riesgo técnico. - Valoración del riesgo de planificación. - Estimación del coste de desarrollo. - Número de elementos físicos incluidos en el modelo de datos.

Tabla 160.Métricas para evaluar la calidad de los modelos Entidad/Relación.Adaptado de [Jose Javier Dorado Cosin]

- Métricas de Kesh

Se desarrolla un modelo, una metodología y métricas para evaluar la calidad del modelo Entidad/relación. La calidad de los modelos Entidad/relación se puede determinar como factores ontológicos y factores de comportamiento. Los factores ontológicos se dividen en dos:

1. Influencia en la estructura: Se refiere a las entidades y sus relaciones, contenido atributos de las entidades.
2. Influencia en el contenido de los modelos entidad/relación.

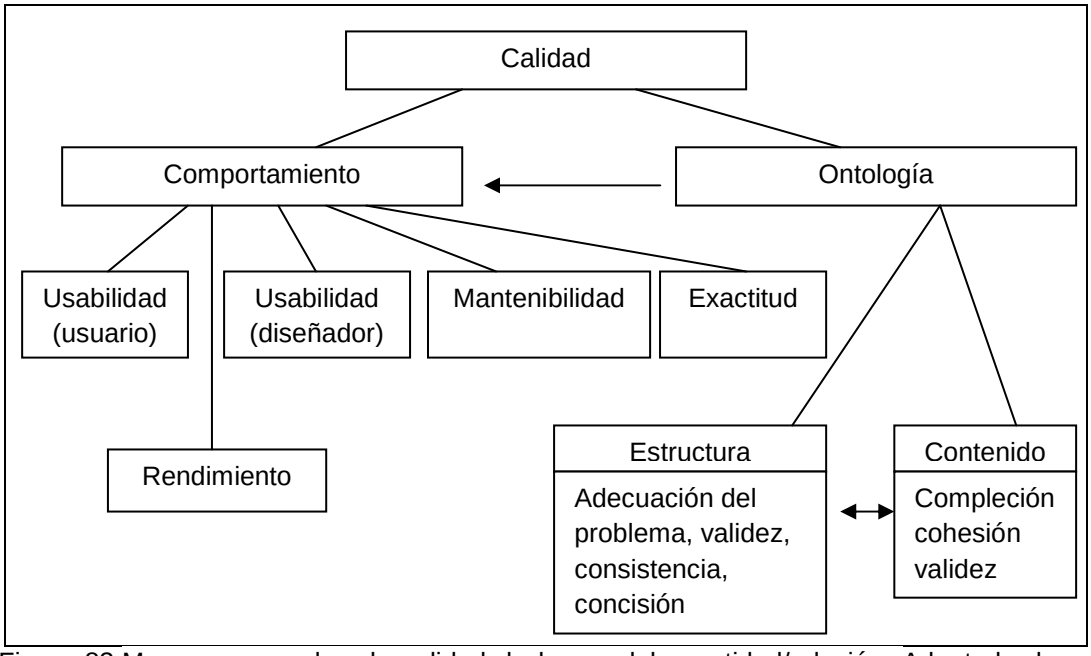


Figura 83.Marco para evaluar la calidad de los modelos entidad/relación. Adaptado de [José Javier Dolado Cosín]

Los factores ontológicos influyen en los factores de comportamiento: estructura, comportamiento.

	Usabilidad (usuario)	Usabilidad (diseñador)	Facilidad de mantenimiento	Exactitud	Rendimiento
Adecuación al problema	X				
Validez		X	X		
Consistencia	X	X		X	
Concisión	X		X		X

Tabla 161.Factores ontológicos referidos a la estructura en los factores de comportamiento. Adaptado de [José Javier Dolado Coisin]

	Usabilidad (usuario)	Usabilidad (diseñador)	Facilidad de mantenimiento	Exactitud	Rendimiento
Compleción	**X**	**X**		**X**	**X**
Cohesión		**X**	**X**		
Validez		**X**			

Tabla 162.Factores ontológicos referidos al contenido en los factores de comportamiento. Adaptado de [José Javier Dolado Coisin]

- Métricas de Eick

  Se plantea para evaluar la calidad de los modelos conceptuales una métrica que se ha integrado dentro de una metodología denominada ANNAPURNA, que es implementada mediante la herramienta CASE. Los esquemas conceptuales de Eick, llamados diagrama S; estos diagramas soportan: clases, conexiones entre clases, atributos, jerarquías de generalización, dependencia en existencia, dependencias funcionales.

- Métricas de Gray: Propone dos métricas para medir y comparar diseños alternativos en el diagrama Entidad/relación.

    1. Complejidad de un diagrama de una entidad.
    2. Complejidad de la arquitectura de los datos para una entidad.

- Métricas de Polo

    Define un marco para medir atributos de calidad en las etapas de diseño conceptual siguiendo la norma ISO [9126- 1998], comprende los siguientes factores:

    1. Compleción: Es una métrica nominal y puede adquirir los siguientes valores: alta, baja o muy baja.
    2. Corrección: Adquiere los siguientes valores: alta, baja o muy baja.
    3. Minimalidad: Se mide en función del número de atributos sobrantes es decir: repetidos o calculables mediante operaciones matemáticas o inferencia lógica.

Métricas para base de datos relacionales

Apenas existen métricas para este tipo de base de datos. El único identificador para medir la calidad de una base de datos relacional es la teoría de la normalización. Se pueden clasificar en dos grupos según la aplicación: métricas orientadas a tablas y métricas a esquemas:

- Métricas orientadas a tablas

    - Número de atributos.
    - Grado de referencialidad.

- Métricas orientada a esquemas

    - Número de atributos.
    - Grado de referencialidad.
    - Profundidad del árbol referencial.
    - Ratio de normalidad.
    - Cohesión del esquema.

Métricas para base de datos objeto-relacionales

Las bases de datos objeto-relacionales mezclan características de las bases de datos relacionales como son: modelo de datos, seguridad, lenguaje de alto nivel, etc. con los principios de la orientación a objetos como son: encapsulamiento, generalización, agregación, polimorfismo etc. Tiene la posibilidad de definir clases o tipos abstractos de datos.

Descripción de datos

La descripción de los datos constituye un aspecto primordial en la especificación del software. Trata de detallar la estructura interna de los datos que maneja el sistema, lo mismo pasa con la descripción funcional.

La descripción de datos no se debe explayar en el diseño o codificación, se describen datos que resulten relevantes para entender lo que el sistema debe hacer. Los datos se pueden describir usando un lenguaje para ello es recomendable usar notaciones específicas, una solución a ello es usar la notación utilizada para definir tipos de datos en aquellos lenguajes estructurales, esta solución tiene como desventaja la dependencia de una sintaxis concreta y la necesidad de detallar aspectos de la fase de diseño o codificación como son el tamaño o el tipo concreto de cada elemento.

La notación adoptada es el diccionario de datos. Esta notación es bastante más informal que cualquier tipo de lenguaje de programación y con ello se logra que la descripción de los datos sea precisa en la especificación de los requisitos.

El diccionario de datos describe el significado de los flujos de datos, almacenes del DFD, los valores de los elementos primarios que definen esos flujos y almacenes y las relaciones entre almacenes descritos en el diagrama entidad/relación. Esto es de gran importancia ya que cualquier persona ajena puede comprender su representación y relaciones que entre ellos exista.
El diccionario de datos al menos tiene que contener la siguiente información para cada dato.

Nombre	Descripción	
Nombre o nombres	Denominación con la que se use el dato en el resto de la especificación. Se puede establecer que para una mejor especificación se usen diferentes nombres para datos que tienen una misma descripción.	
Utilidad	Se indica procesos, almacenes de datos, descripciones funcionales, almacenes de datos, etc., en lo que se use el dato.	
Estructura	Se indican los elementos de los que está constituido el dato y usa la siguiente notación:	
	A+B	Secuencia o concatenación de los elementos A+ B.
	[A | B]	Selección entre los distintos elementos A o B
	{ A } ^{N}	Repetición N veces del elemento A, se suprime N si es indeterminado.
	(A)	Opcionalmente se podrá incluir el elemento A.
	/ descripción /	Comentarios de la descripción de un lenguaje.
	=	Separador entre el nombre de un elemento y su descripción.

Tabla 163.Diccionario de datos

Diagrama de modelado de datos

En los sistemas de información se maneja una gran cantidad de datos que están relacionados entre sí, lo cual implica la necesidad de establecer una organización para facilitar operaciones que se quieren desarrollar con ellos. La necesidad de establecer en una organización los datos es para que se simplifique y agilice su uso, esta organización configura la estructura de la base de datos que utilizará el sistema.

Es primordial que en la especificación se establezca el modelo de datos que se considera el más oportuno para conseguir todos los requisitos del sistema. Este modelo se conoce como modelo entidad/relación y permite definir los datos que manejará el sistema junto con las relaciones que se desea que existan entre ellos. Este modelo se revisará durante las fases de diseño y codificación como sucede con el resto de la especificación. Es el punto de inicio indispensable para comenzar cualquier diseño. Para ello vamos a realizar un estudio de modelo entidad / relación.

4.3.7.6.2 Elementos de diseño

Diseño de entidades: Los elementos de datos, tipos de datos y las clases, almacenes de datos y mecanismos de acceso.

- Diagrama de flujo de datos (DFD)

  Es una técnica gráfica que representa el flujo de información y transformaciones que se aplican a los datos al moverse desde la entrada a la salida.

  Componentes de un DFD

  - Entidad externa: Muestras entidades externas en las cuales se comunica el sistema. Pueden ser: Personas, lugares, cosas, organizaciones u otro sistema con el cual se relaciona.
  - Proceso: Parte del sistema que transforma unos datos de entrada en salida.
  - Flujo de datos: Indica la dirección de los datos que entran y salen de un proceso.
  - Almacén de datos: Sirve para modelar un conjunto de datos en reposo es decir ficheros o registro en la memoria. Es utilizado por varios procesos.

  Un DFD se va refinando en niveles, representa un mayor flujo de información y un mayor detalle llamado nivel de contexto que representa el sistema completo, después se explosiona el proceso que hay en el nivel de contexto para pasar al primer nivel, estos procesos del nivel 1 se enumeran.

Los procesos del nivel 1 se vuelven a explosionar en procesos más pequeños y se vuelven a enumerar de forma consecutiva. El refinamiento de los DFD continúa hasta que cada proceso represente una manera sencilla que se pueda implementar en cualquier lenguaje de programación.

Cuando se explosionan los niveles hay que tener en cuenta:

1. Cada proceso se debe dividir en un nivel al siguiente no más de seis procesos.
2. No todos los procesos del nivel 1 se pueden explosionar en el mismo número de niveles es decir un proceso se puede explosionar en más niveles que otros.
3. Los DFD han de ser coherentes entre sí, es decir los flujos de datos que entran y salen de un proceso en un nivel específico se deben corresponder con los que entran y salen de los procesos inferiores.

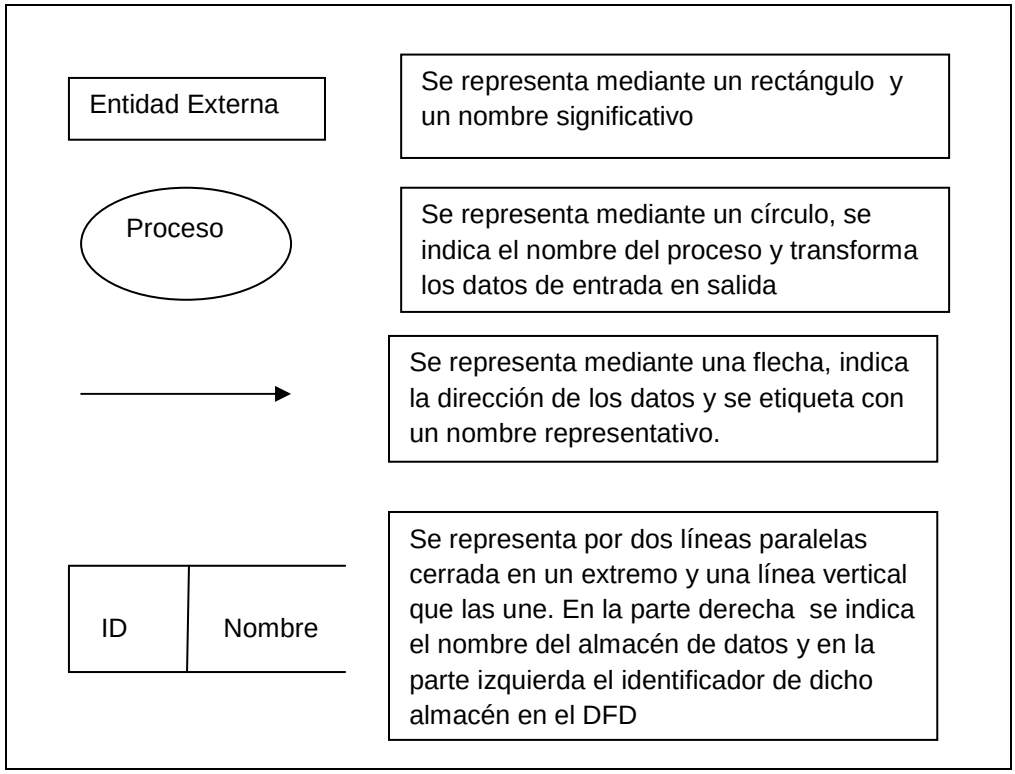


Figura 84.Diagrama de DFD

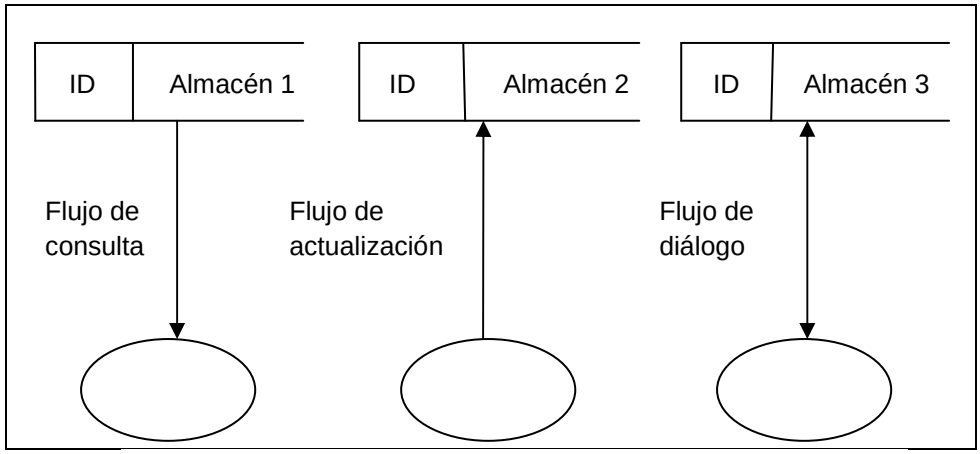


Figura 85.Representación de los flujos de datos entre procesos y almacenes

- Diagrama entidad/relación

  El enfoque en el diagrama entidad/relación es tratar de proporcionar un punto de vista de datos en el modelado de un sistema. Al igual que con muchas anotaciones que pueden ser utilizadas tanto en el análisis del problema y durante el desarrollo de la solución o sea el diseño.

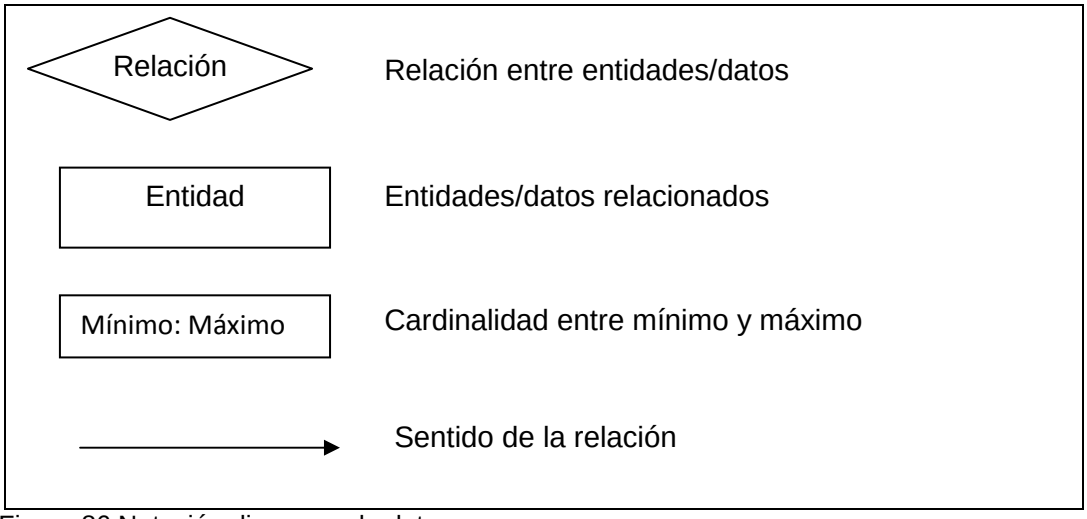


Figura 86.Notación diagrama de datos

El diagrama entidad/relación, se utiliza principalmente para captar las relaciones que existen entre los objetos de datos estáticos en un modelo de problema o un modelo de diseño.

En particular, el modelo de entidad-relación ha proporcionado una base fundamental para el desarrollo de los modelos relacionales que se utilizan en los sistemas de base de datos. El diagrama entidad/relación puede desempeñar otras funciones además de su uso en el diseño de base de datos. Por ejemplo, a veces se utiliza para modelar la forma detallada de los datos almacenados en un sistema. En una escala más grande se puede utilizar para modelar las relaciones muy complejas y diseño abstracto "objetos".

Los componentes de un diagrama entidad/relación son:

- Tipos de objetos: Se representa por medio de una caja rectangular y específica un conjunto de objetos cuyas instancias se identifica de una manera única, juegan un papel primordial en el sistema y describen una serie de datos.

- Relaciones: Representa la conexión entre sí mediante relaciones, su representación gráfica es mediante un rombo. La relación representa conexiones y cada objeto (instancia) representa una relación de asociación entre una ocurrencia de la instancia y otra ocurrencia del otro.

  En el diagrama entidad/relación las relaciones son multidireccionales es decir que se puede leer en cualquier sentido y no precisan mostrar la cardinalidad es decir, no muestran el número de objetos que participan en la relación. La cardinalidad representa instancias de un objeto que se conectan con otro es decir el número mínimo y el número máximo. Ejemplo de representación: número mínimo ":" número máximo.

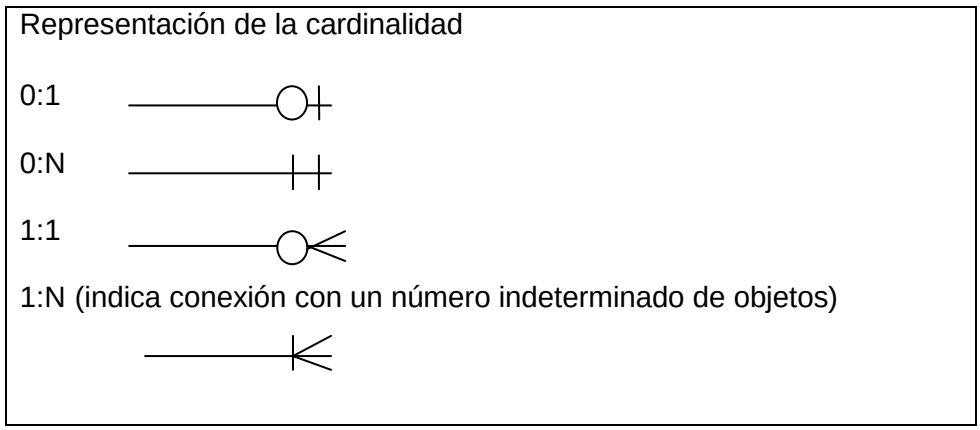


Figura 87.Significado de cardinalidad

- Indicadores asociativos: Funcionan como objeto y relación. También el indicador asociativo representa una relación que se desea conservar alguna información.

- Indicadores supertipo/subtipo: Consiste en dos tipos de objetos conectados mediante una relación; el supertipo representa una categoría más general y el subtipo representa una categoría específica. Sirven para representar jerarquía de objeto, la representación del supertipo en la parte superior de la jerarquía conectado a través de la relación con una línea que contiene una barra y en la parte inferior los distintos subtipos.

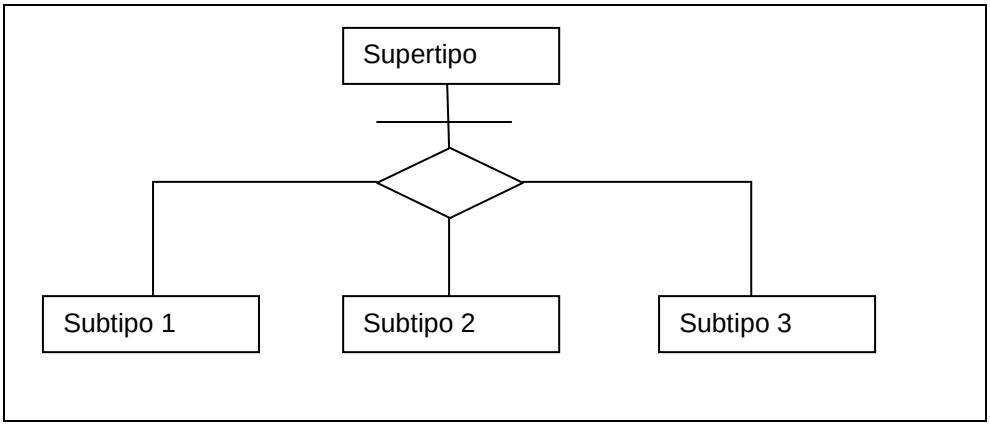


Figura 88.Indicador de supertipo – subtipo de una jerarquía

Relaciones de diseño

Partiendo del modelo entidad/relación. Para obtener los esquemas de las tablas de una base de datos relacional, reflejan la visión lógica de los datos y que sean eficaces. En el modelo relacional la eficacia se contempla desde varios puntos de vista:

- Formas normales: Establecen esquemas de tablas a través de criterios para que sean claros y que no haya redundancia. Las formas normales:

    - Primera forma normal: una tabla está en primera forma normal, si la información asociada a cada una de las columnas es un valor único, y no hay colección en un número variable.
    - Segunda forma normal: una tabla está en segunda forma normal, si está en primera y además tiene una clave primaria que distingue cada fila y cada casilla que no sea de clave primaria y depende de toda la clave primaria.
    - Tercera forma normal: una tabla se encuentra en tercera forma normal, si se cumple la segunda forma normal y cada valor de la columna que no es clave primaria depende directamente de la clave primaria esto quiere decir no hay dependencias entre columnas que no son clave primaria.

- Entidades de diseño: En el modelo relacional cada entidad del modelo entidad/relación se traduce en una tabla por cada clase de entidad, con una fila por cada elemento de la entidad y una columna por cada atributo de esa entidad. Si una entidad está relacionada con otras, y se quiere tener una referencia entre entidades que se correspondan una con otra se puede añadir una columna que contenga un código o número de referencia que identifique cada elemento de datos.

- Relación de asociación: Depende de la cardinalidad de la relación para almacenar en tablas la información. Traducir la relación a una tabla conteniendo referencias a las tablas de las entidades relacionadas así como los atributos de la relación si los hay. Esto es correcto para relaciones con cualquier tipo de cardinalidad.

- Relación de composición: Las relaciones de composición o agregación se tratan de la misma forma que las relaciones de asociación. En la relaciones de composición la cardinalidad del objeto compuesto por lo general es 1.

- Relación con jerarquía: Cuando una entidad genérica tiene varias entidades especificas se puede optar de tres maneras distintas para almacenar la información en tablas:

    - Tabla para la superclase, con los atributos comunes heredados por la subclase, y otra tabla por cada subclase con sus propios atributos.
    - Si se han repetido los atributos comunes en cada subclase, esto implica que la tabla de la superclase desaparece.
    - Se prescinde de las tablas separadas por cada subclase y se extiende la tabla de superclase con todos los atributos que contienen las subclases en forma de campos o columnas con valores opcionales.
    - Diseño de índices: Se accede a un dato concreto reduciendo el tiempo de acceso. A cambio de aumentar el espacio de almacenamiento, aumenta el tiempo para almacenar cada nuevo dato y para modificar el valor de un atributo indexado.

Atributos de diseño: Las propiedades de persistencia y calidad.

Se denomina clase persistente a las clases que pueden contener objetos persistentes y clases temporales no persistentes.

Relación de un objeto persistente: se lleva a cabo mediante dos operaciones:

1. Leer: es necesario en la práctica poder identificar y borrar entre objeto persistente de la misma clase.

2. Grabar: de un objeto se graban los valores de los atributos. Puede ser que no todos los atributos de un objeto persistente sean persistentes.

Podemos distinguir tres tipos de almacenamiento de manera como se implemente la persistencia:

- Bases de datos orientadas a objetos

    Es el caso más sencillo de todos, esto implica que no hay que transformar los objetos para hacerlos persistentes. Para especificar que los objetos de una clase pueden ser persistentes solo con la definición de clase, apuntando qué atributos son persistentes y qué política de lectura de objeto se requiere. A partir del concepto de clase de esta manera se genera el código que pasa a un procesador para agregar los métodos que gestionan la persistencia de los objetos.

- Bases de datos relacionales y archivos clásicos

  El modelo para bases de datos relacionales y archivos clásicos, a un objeto le corresponde una fila de una tabla de una base de datos relacionales o un registro de archivo y los atributos de los objetos le corresponden las columnas. Antes de grabar un objeto es necesaria la transformación. Por lo tanto es necesario realizar la transformación inversa antes de que se use una vez se haya leído. Existen tres maneras en la gestión de la persistencia para llevar a cabo el sistema de almacenamiento:

  1. Una ventaja importante es que cada clase persistente tenga operaciones para que sus objetos se lean y graben, esto lo hace eficiente con respeto a otras, esto implica menos llamadas de objetos. Tiene una desventaja, la implementación de las clases persistentes que dependen de un sistema de gestión de bases de datos por el simple hecho de que son clases de entidades que pertenecen al dominio del software lo cual restringe la portabilidad.

  2. El segundo método define una clase llamada gestor de disco para cada clase persistente. El gestor de disco accede al fichero o base de datos. Con este método, la clase de entidades se desacopla del sistema de gestión de base de datos esto proporciona una ventaja al gestor de disco, puede hacer de memoria cache de los objetos de la clase de entidades, es decir los objetos se materializan dentro de instancias del gestor antes de crear instancias de la clase de entidades pero resulta menos eficiente que el primer método.

  3. Tercer método es una combinación del primer y segundo método consiste en crear los gestores de disco y permite añadir operaciones de lectura, grabar etc., a las clases del dominio. La diferencia del primer método es que las operaciones de clases no implementan la persistencia directamente, sino que realizan las llamadas al gestor de disco.

  Para obtener la definición de la estructura de bases de datos relacional o archivos se realizan los siguientes procesos:

  1. Transformar el modelo estático en un modelo de entidad-relación:
     Es muy parecido al modelo orientado a objetos ya que soporta el tipo de entidades. La trasformación del modelo estático al modelo entidad/relacional consiste en convertir cada clase con un atributo con valores múltiples en dos clase unidas por medio de una asociación y después hacer corresponder un tipo de entidad a cada clase no asociativa.

  2. Suprimir la herencia, el modelo entidad/relación no la soporta. Por lo tanto se puede resolver esta situación de la siguiente manera:

- Definición de cada clase en una tabla; ésta dispone de los atributos propios de su clase como los heredados.
- Creación de una tabla para la superclase y una complementaria para cada subclase que contiene atributos propios. Cada tabla complementaria contendrá los atributos propios de la subclase más los atributos que identifican cada fila es decir cada objeto. En la tabla de la superclase estarán los atributos heredados y no heredados de cada objeto de la subclase.
- Crear una tabla para la jerarquía de clases que contendrán los atributos propios de cada subclase más lo de la superclase.
  Como podemos observar esto crea inconvenientes:

  - Hay tantas clases como subclases.
  - Creación de un gestor de disco para cada subclase.
  - Complica los procesos relacionados con el cliente, esto implica hacer una fusión de todas las tablas.

3. Transformar el modelo entidad/relación en un modelo relacional según las reglas del diseño de bases de datos relacionales. Los archivos clásicos en lugar de tablas relacionales tendrá archivos planos.

4. Creación de un gestor de disco para cada clase persistente que implemente los accesos a las tablas o archivos planos correspondientes.

5. Base de datos objeto-relación

   El método diseño de la persistencia de la base de datos objeto-relación, es el mismo que la base de datos relacional pero con la diferencia de que este modelo objeto-relación puede contener atributos tipo compuesto, esto implica la no creación de gestores de disco para las clases persistentes.

4.3.7.6.2.1Los datos de los atributos

Una descripción de los elementos de datos internos de la entidad. El atributo de datos describe el método de representación, los valores iniciales, el uso, la semántica, el formato y los valores aceptables de los datos internos.

En el proceso unificado, el glosario también juega el rol de diccionario de datos, es un documento que recoge los datos sobre los datos, es decir metadatos. Los atributos de los términos pueden contener:

- Alias.
- Descripción.

- Formato (tipo, longitud, unidad).
- Relaciones con otros elementos.
- Rango de valores.
- Reglas de validación.

Estructura de datos

Nos referimos a las diferentes maneras que emplean los lenguajes para estructurar los datos que manejan:

- Datos simples

  Los lenguajes de programación manejan datos de tipo entero incluyendo el rango que puede ser: rango normal, rango corto, rango largo, etc. De igual manera los datos reales se manejan en lenguaje de programación teniendo en cuenta su precisión, se dispone de dos formas de precisión: precisión simple y precisión doble.

  Lenguajes que desarrollan cálculo científico pueden utilizar datos más complejos. Hoy en día los lenguajes de programación utilizan datos de tipo carácter empleando el código ASCII; la mejor manera de utilización de los datos es estudiar la forma específica de cada lenguaje empleado. Los datos de tipo subrango permiten acotar el rango de tipo ordinal para crear un nuevo tipo de dato.

  Este tipo de dato es de gran importancia en los lenguajes que los soportan por la facilidad de depuración y el mantenimiento de programas y dispone de una detección automática en tiempo de ejecución de los valores que no se encuentran en el rango establecido.

- Datos compuestos

  Depende del compilador o de la versión estándar del lenguaje. Los datos compuestos se definen como la combinación de otros datos simples y compuestos ya definidos. Los lenguajes de programación definen y utilizan matrices en las cuales cada lenguaje de programación debe familiarizarse con la sintaxis y semántica. Los datos compuestos de elementos heterogéneos se desarrollan en la mayoría de los lenguajes de programación mediante la estructura registro.

- Constantes

  En los primeros lenguajes de programación se emplearon constantes literales ya sea por su valor numérico o de caracteres. Los lenguajes de programación modernos se declaran constantes simbólicas, con un nombre. Los valores simbólicos se evalúan en tiempo de ejecución, esto permite una inicialización dinámica de las constantes.

- Comprobación de tipos: se distinguen los siguientes niveles:

    1. Nivel 0: Sin tipos; pertenecen los lenguajes en los que no se pueden declarar nuevos tipos de datos y todos los datos deben pertenecer a los tipos predefinidos.

    2. Nivel 1: Tipado automático; el compilador es el encargado de decidir el tipo de dato más apropiado que usa el programador. El compilador tiene como función convertir al tipo de dato los operandos de una expresión cuando no son compatibles entre sí o cuando lo son con el operador usado en la expresión

    3. Nivel 2: Tipado débil: Se realiza una conversión automática de tipos entre datos que poseen ciertas características iguales. La conversión automática de valores lógicos a numéricos o viceversa no es factible pero sí son posibles las conversiones entre enteros y reales. En expresiones con operando de diferentes tipos ya sean enteros, reales, etc., las conversiones se realizan hacia el tipo de dato de mayor rango o precisión.

    4. Nivel 3: Tipado semirrígido: El lenguaje Pascal es el más representativo en este nivel. Los datos que se quieren utilizar deben ser declarados antes con sus correspondientes tipos. Los tipos de datos son incompatibles con los que se declaran por separado con nombres distintos aunque posean la misma estructura. No se pueden desarrollar operaciones entre datos de tipos incompatibles. La comprobación del número de argumentos y el tipo de cada función y procedimiento para que coincidan la declaración con su uso.

    5. Nivel 4: Tipado fuerte: Las comprobaciones de tipos de datos se realizan en compilación, carga y ejecución. El programador está obligado a realizar cualquier conversión que necesite usar.

4.3.7.7 Punto de vista de los patrones de uso

Este punto de vista sobre ideas de diseño (conceptos emergentes) como patrones de colaboración implican funciones abstractas y conectores.

El concepto del patrón de diseño está muy asociado con el estilo de arquitectura orientada a objetos, aunque en principio no hay razones para que los patrones no puedan trabajar con otros estilos. El estilo: arquitectura orientada a objetos, es la que ha hecho necesario el uso más complejo y por lo tanto menos accesible a los métodos de diseño, dando lugar a una búsqueda para encontrar formas alternativas de desarrollo de diseños.

4.3.7.7.1 Consideraciones de diseño

Los asuntos clave incluyen la reutilización en el plano de las ideas de diseño (patrones de diseño), los estilos arquitectónicos, y las plantillas de marco.

Realizaremos un estudio de la descripción de programación orientada a objetos.

Un objeto desde el punto de vista, es un desarrollo evolutivo del concepto de tipo abstracto de datos, que proporciona una abstracción de una parte-solución que:

- Tiene un estado que se resume en el objeto.
- Posee un comportamiento en que responde a los acontecimientos externos.
- Posee identidad.

La inspección o la modificación de su estado solo puede obtenerse mediante la utilización de los métodos externos que son proporcionados por el objeto a la interfaz con el mundo exterior.

La principal característica de un objeto facilita una abstracción parcial de la solución. Integra la noción de encapsulamiento de la información de estado. Por lo tanto un enfoque de diseño orientado a objetos es la identificación de estos objetos del mundo real. Para la mayoría de los propósitos, sin embargo, los patrones de diseño se han centrado en la descripción de objetos de su aplicación, que proporcionan elementos reutilizables de soluciones en lugar de la descripción de los objetos del problema.

Dentro de un catálogo de patrones de diseño se pueden clasificar de varias maneras, según Gang of Four (Banda de los cuatro).

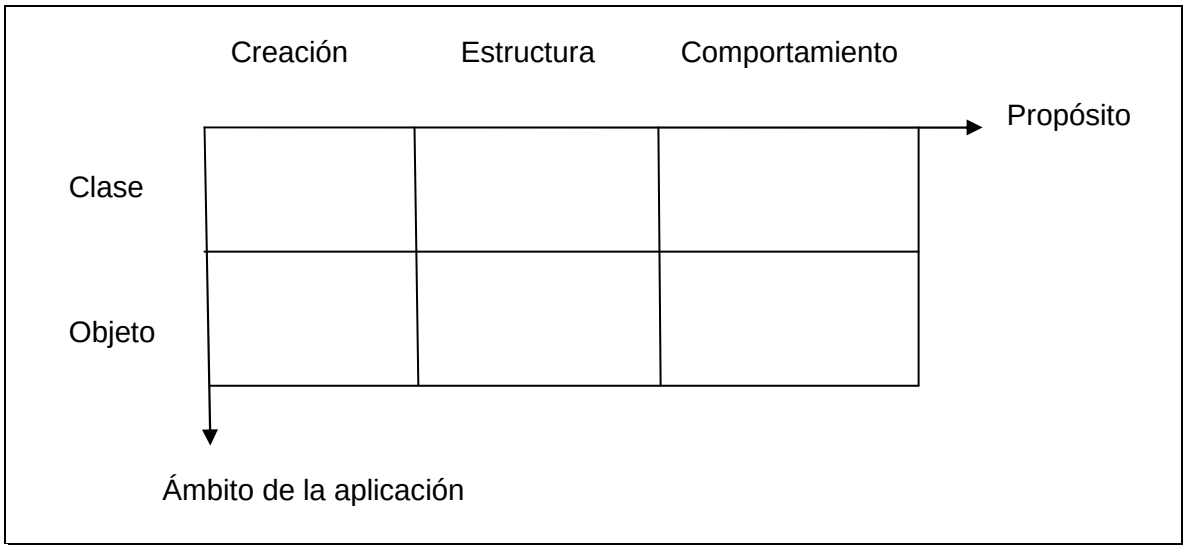


Figura 89.Patrón de esquema de clasificación utilizado por Gang of Four (GoF)

1. Propósito: Finalidad de lo que un patrón de diseño utiliza, generalmente se describe se la siguiente manera:

    - Patrones de creación: Creación de objetos. Se utiliza para la instanciación con el propósito de hacer que el software sea independiente, como las clases y objetos se crean, representan y agregan. Los patrones en el nivel de clases utilizan herencia para variar la clase y a nivel de objetos se delega la instanciación a otro objeto.

    - Patrones estructurales: Clases y objetos en su composición, se combinan para formar estructuras mayores. En el nivel de clase se utiliza la herencia para agregar interfaz o implementación, en el nivel de objetos la agregación se hace en tiempo de ejecución.

    - Patrones de comportamiento: Describe la forma en que las clases u objetos interactúan y cómo la responsabilidad se asigna entre ellos. En el nivel de clases se aplica a la herencia y el nivel de objetos la agregación.

2. Ámbito de la aplicación: Describe si el patrón de diseño trata el uso de las clases u objetos que podemos considerar como las clases de plantillas de los objetos que se crean instancias es decir:

    - Patrones sobre clases: Relaciones de herencia, se establecen en tiempo de compilación.
    - Patrones sobre objetos: relaciones entre objetos, pueden variar dinámicamente en tiempo de ejecución.

3. Clasificación por niveles:

    - Patrones arquitectónicos: Relativos a la forma de todo el sistema. Tratan sobre la estructuración de un sistema de software en subsistemas.
    - Diseño: Patrones de diseño que se relacionan con la forma detallada de los subsistemas. Patrones dirigidos a solventar problemas de diseño.
    - Frases Hechas: Describen la manera de implementar algo. En lenguajes de programación describen la estructura general del código.

Para el caso de los patrones de diseño que adoptan un sistema de agrupación de patrones basadas en roles.

- Descomposición estructural: Patrones de apoyo, una descomposición adecuada de los subsistemas y componentes complejos en partes cooperantes.
- Organización del trabajo: Tiene que ver con la colaboración de componentes.

- Control de acceso: Describe los patrones que protegen y controla el acceso a los servicios o componentes.
- Gestión: Patrones de dirección de la organización de las colecciones de objetos.
- Comunicación: La comunicación entre los componentes del sistema.

Podemos contemplar un importante patrón de arquitectura denominado patrón por capas: representa uno de los paradigmas actuales de desarrollo de aplicaciones. Estructura el sistema en las principales capas. Se basa en ideas:

- Estructura lógica, organiza un sistema en capas separada de responsabilidades diferentes y relación, con una separación clara: los niveles más bajos son de servicio general y los niveles más altos son más específicas de la aplicación.
- Colaboración y acoplamiento se desarrollan desde las capas más altas a las capas más bajas, evitando el acoplamiento desde las capas inferiores a las capas superiores.
- Cada capa a menudo está compuesta de subsistemas o paquetes, describe la organización conceptual, independientemente del empaquetamiento o despliegue físico.

El patrón por capas define un modelo N-niveles de arquitectura lógica en capas, como rentabilidad de organizar el software de esta manera permite disminuir el efecto de propagación de cambios eludiendo que la capa de la interfaz se mezcle con la lógica de la aplicación. El objetivo y número de capas cambie de una aplicación a otra y entre dominios de aplicación como pueden ser: sistemas operativos, sistemas de información etc.

Algunos patrones de diseño más conocidos se describen a continuación:

- Nombre: Nombre del patrón.

- Tipo: Puede ser: Creación, estructura, arquitectónico, comportamiento.

- Sistema:

  - Gamma, Helm, Johnson y Vlissides. Llamada Banda de los Cuatro o GoF , (GHJV)
  - Buschmann, Meunier, Rohnert, Sommerland y Stal.(BMRSS)
  - Patrones documentados por Larman, corresponden a principios de diseño en términos generales más que a problemas específicos. (Larman)

- Propósito: Alcance de cada patrón de diseño.

Nombre	Tipo	Sistema	Propósito
Abstract Factory	Creación	GHJV	Crea objetos relacionados mediante una interfaz sin que sea oportuno conocer su clase.
Adapter	Estructura	GHJV	Sustituye la interfaz de una clase esto conlleva la reutilización de algunas clases
Builder	Creación	GHJV	Puede crear distintas representaciones bajo la construcción de un objeto complejo.
Comand	Comportamiento	GHJV	Transforma una llamada de una operación en un objeto manejable.
Composite	Creación	GHJV	Construye una jerarquía de composición de objetos.
Creator	Estructura	Larman	Asigna a la clase B la responsabilidad de crear los objetos de otra clase A si B ya agrega, contiene, registra o utiliza de una manera principal objetos de A o tiene información para inicializarlos.
Chain of responsability	Comportamiento	GHJV	Implementa llamadas de operaciones por medio de otros objetos.
Blackboard	Arquitectónico	BMRSS	Coordina la comunicación entre los componentes del software distribuido.
Controller	Comportamiento	Larman	Asigna el tratamiento de los sucesos a las clases que representan el sistema o la organización.
Microkernel	Arquitectónico	BMRSS	Dentro de un software, separa el núcleo funcional de la funcionalidad añadida.
Layers	Arquitectónico	BMRSS	Estructura una aplicación en grupos de distintos niveles en subtareas.
Iterator	Comportamiento	GHJV	Accede a los componentes de un objeto agregado, sin ver su información.
Proxy(1)	Estructura	GHJV	Controla el acceso. Proporciona una representación para un objeto que no hay que crearlo hasta que se emplee.
Proxy(2)	Estructura	BMRSS	Los clientes de un componente piden las operaciones a un representante de este y no a él directamente por razones de eficacia.
Singleton	Creación	GHJV	Asegura que en una clase existe un objeto.
State	Comportamiento	GHJV	Cuando un objeto cambie de estado que cambie su comportamiento como si hubiera cambiado de clase.
Strategy	Comportamiento	GHJV	Encapsula varios algoritmos haciéndolos intercambiables

Visitor	Comportamiento	GHJV	Sustituye la interfaz de una clase, permitiendo reutilizar algunas clases.

Tabla 164.Descripción detallada de algunos patrones de diseño adaptada [Benet Campderrich Falgueras]

4.3.7.7.2 Elementos de diseño

Entidades de diseño

Clase

La clase especifica los datos, la implementación de un objeto queda definida a través de la clase, la definición de las operaciones que pueden desarrollar y la representación interna de un objeto. Los objetos crean la instancia de una clase.

La creación de una instancia de una clase asigna espacio de almacenamiento para los datos internos del objeto y relaciona las operaciones con esos datos. La flecha discontinua representa que una clase crea objetos de otra clase, es decir la flecha apunta a la clase de los objetos creados.

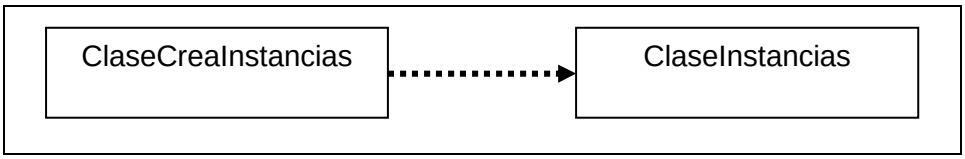


Figura 90.Representación clase que crea instancias a clase instanciada

Las nuevas clases se definen en términos de existencia es decir utilizando herencia. Una subclase hereda de una clase padre datos y operaciones de la clase padre. Los objetos que son instanciados de las subclases tendrán todos los datos definidos por la subclase y por la clase padre y desarrollarán las operaciones definidas por las subclases y la clase de sus padres.

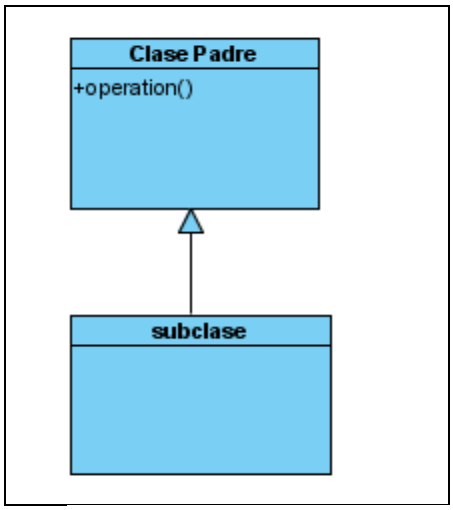


Figura 91.Representación herencia de clases

Clase Abstracta

Define una interfaz común para las subclases. La clase abstracta delegará parte o toda su implementación en las operaciones que están definidas en las subclases. La clase abstracta define operaciones pero no se implementan por lo tanto se denominan operaciones abstractas y aquellas clases que no son abstractas se denominan clases concretas. La herencia de clases define clases ampliándolas a otras clases, esto implica facilidad de definir familias de objetos de parecida funcionalidad.

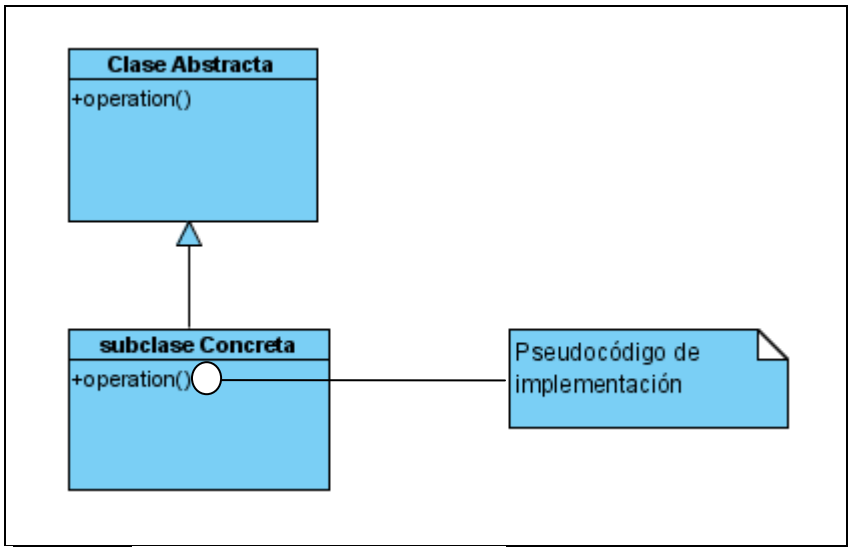


Figura 92.Representación clase abstracta

Clase mezclable: Parece una clase abstracta en que no está pensada para la creación de instancias de ella, proporciona interfaz opcional a otras clases. Soporta herencia múltiple.

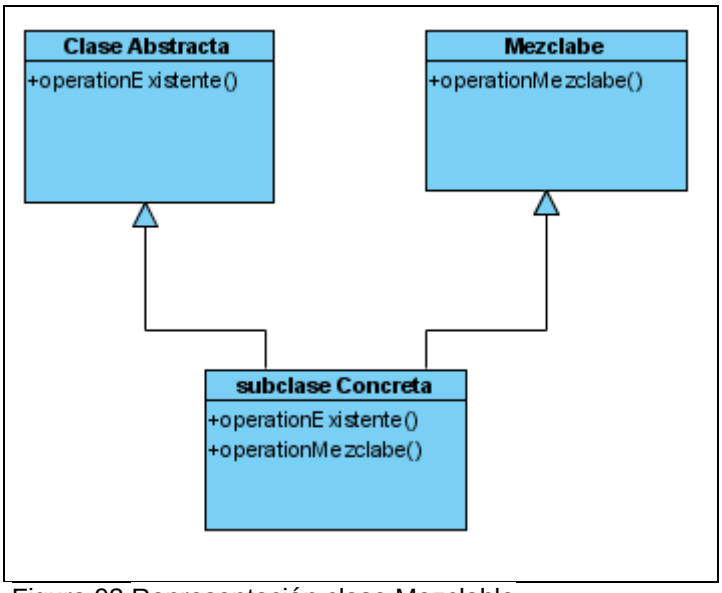


Figura 93.Representación clase Mezclable

Herencia de clases versus herencia de interfaces

La clase de un objeto se define como se implementa un objeto. La clase define el estado interno del objeto y por consiguiente sus operaciones. El tipo de un objeto se refiere a la interfaz. Un objeto puede tener tipos y objetos de clases distintos que pueden tener el mismo tipo.

La herencia de clases y herencia de interfaz o subtipo tiene como diferencia que la herencia de clases define la implementación de un objeto, es decir es un mecanismo que comparte código y representación. La herencia de interfaces o subtipo describe como se puede utilizar un objeto en vez de otro. Es necesario hacer énfasis en esta diferencia ya que muchos lenguajes de programación no hacen esta distinción.
Muchos de los patrones de diseño dependen de esta diferencia. Por ejemplo:

- Objetos en el patrón cadena de responsabilidad, debe tener un tipo común pero no comparten la misma implementación.
- El patrón composite, el componente define una interfaz común y el compuesto define una implementación común.
- Los patrones observer, strategy, state y command pueden implementarse con clases abstractas que son interfaces puras.

Herencia frente a composición

La herencia de clases y la composición de objetos, son dos técnicas para reutilizar funcionalidad en sistemas orientados a objetos. La reutilización mediante la herencia se denomina reutilización de caja blanca es decir visibilidad ya que las clases padres son visibles a las clases hijas.

La composición de objetos es una alternativa a la herencia de clases y requiere que los objetos a componer tengan interfaces definidas a este concepto se le denomina reutilización de caja negra ya que los detalles de sus objetos internos no son visibles.

Nombre	Ventajas	Inconvenientes
Herencia de clases	Se define estáticamente en tiempo de compilación. Facilidad de uso. La implementación que está siendo reutilizada es más fácil su modificación	No permite la modificación de las implementaciones heredadas de las clases padres en tiempo de ejecución. Las clase padres definen al menos parte de la representación de las clases hijas. Dependencias e implementación suelen causar fallos al tratar de reutilizar una clase hija, la dependencia limita la flexibilidad y reutilización. Como solución es aconsejable utilizar clases abstractas

Tabla 165.Ventajas e inconvenientes de la herencia de clases

Composición de objetos

- Se definen dinámicamente en tiempo de ejecución.
- Cualquier objeto puede ser cambiado en tiempo de ejecución por otro de manera que sean del mismo tipo.
- Los objetos acceden a través de sus interfaces en las cuales no se rompe su encapsulación.
- La dependencia e implementación son más pequeños, ya que un objeto se trata en términos de interfaces de objetos.
- La composición de objetos frente a la herencia de clases ayuda a conservar cada clase encapsulada y centrada en una sola tarea.

Delegación

Es una manera de lograr que la composición sea eficaz en la reutilización como lo es en la herencia. Su principal ventaja es que hace que sea fácil mezclar comportamientos en tiempo de ejecución y modificar la manera en que estos se mezclan y como inconveniente, el software dinámico y parametrizado es más complicado de comprender que el software estático. Muchos patrones de diseño utilizan la delegación.

- Patrón State: Un objeto delega peticiones en un objeto estado que representa su estado actual.
- Patrón Strategy: Un objeto delega peticiones en un objeto que representa una estrategia para llevarla a cabo.
- Patrón Visitor: La operación se realiza sobre cada elemento de una estructura de objetos siempre se delega en el objeto visitante.

Herencia versus tipos parametrizados

Los tipos parametrizados se conocen también como genéricos o plantillas. Sirven para reutilizar funcionalidad. Permite definir un tipo sin especificar los demás tipos que se están utilizando. Los tipos sin especificar proporcionan parámetros cuando se va a usar el tipo parametrizado y permite cambiar los tipos que puede usar una clase.

Utilización de patrón

El diseñador debe adquirir conocimientos de los distintos patrones y familiarizarse con cada uno de ellos que le permitan reconocer situaciones para ser utilizados. Se describe la manera de utilizar un patrón:

- *Hacer una lectura general de los puntos de la documentación del patrón que no se hubiese leído todavía.*
- *Estudiar con detalle los apartados sobre participantes, estructura y papel de los participantes.*
- *Estudia el ejemplo resuelto.*

- *Reemplazar la terminología abstracta del patrón por la del proyecto y se establece una lista de equivalencias*
- *Se definen nuevas las clases, y se modifican las ya existentes que haga falta entre las que ya se habían definido durante el proyecto.*

Sería conveniente clasificar un problema determinado, de igual manera que se clasifican los patrones de diseño para identificar un conjunto de patrones potencialmente útiles. Se pueden resumir de la siguiente manera:

1. Especificar el problema de igual manera que los subproblemas que se encuentran involucrados.
2. Seleccionar en que categoría se encuentra el patrón para que se ajuste a la necesidad del diseño que se trate, en el cual puede ser patrones arquitectónicos o de diseño.
3. Seleccionar la categoría de problema de forma adecuada al problema.
4. Comparar las características del problema con el sistema disponible de los patrones para seleccionar la que más se adecue al problema.
5. Comparar los beneficios para evaluar el diseño de soluciones de compromiso.
6. Seleccionar la variante que se ajuste a patrones más que al problema.
7. El reciclaje repetitivo de los pasos anteriores no debe coincidir con encontrar cualquier patrón existente.

Estos pasos se describirán mediante la siguiente figura:

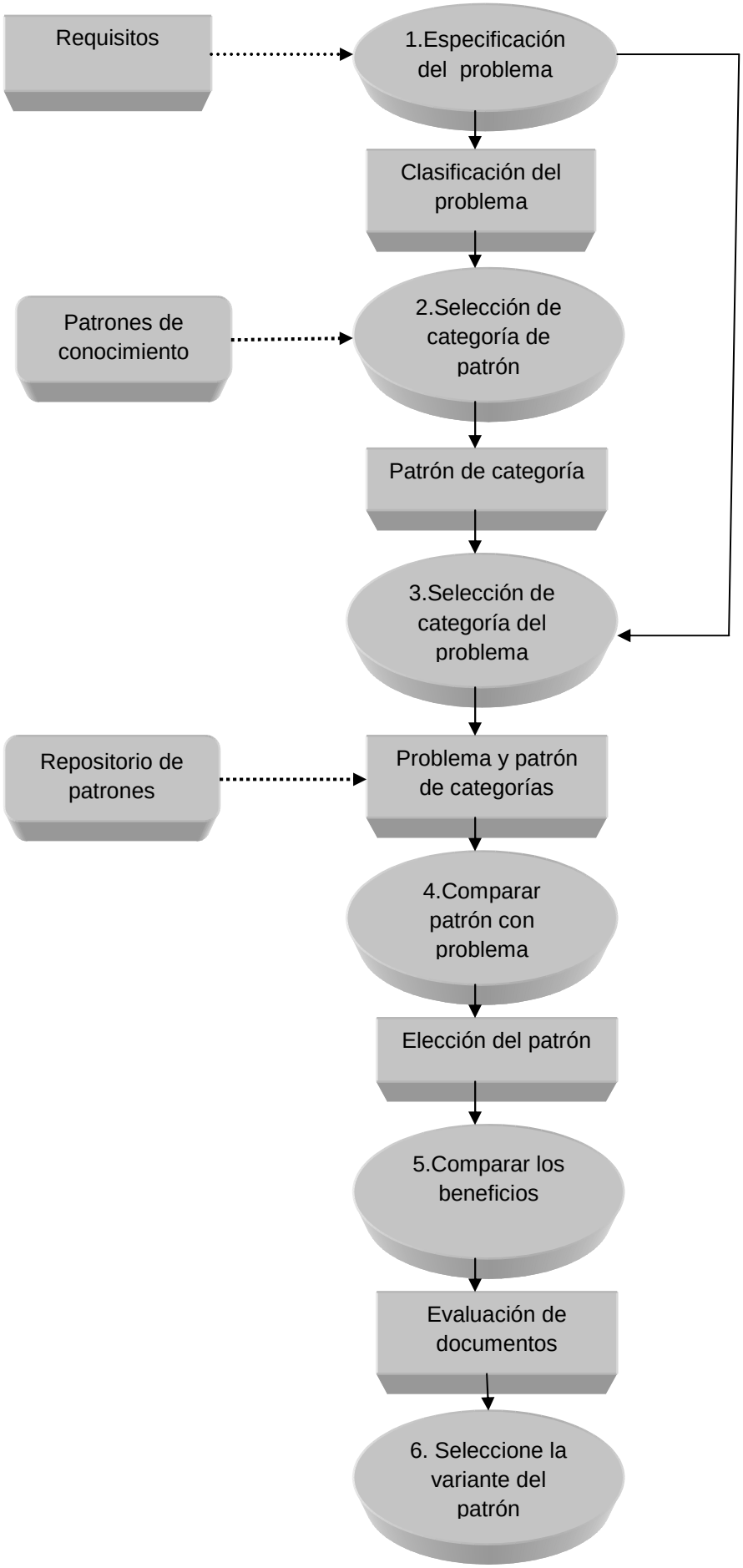

Requisitos
1.Especificación del problema
Clasificación del problema
Patrones de conocimiento
2.Selección de categoría de patrón
Patrón de categoría
3.Selección de categoría del problema
Repositorio de patrones
Problema y patrón de categorías
4.Comparar patrón con problema
Elección del patrón
5.Comparar los beneficios
Evaluación de documentos
6. Seleccione la variante del patrón

Figura 94.Modelo de proceso para el diseño con los patrones

Esta estructura de clasificación incluye estilos arquitectónicos, patrones de diseño y lenguajes en la cual el proceso anteriormente mencionado podría ser empleado desde alto nivel hasta diseño detallado de programas, podemos observar que implica una utilización de estrategia de arriba hacia abajo ya que puede ser inesperado dado que los patrones tienen un concepto de composición.

Se aconseja al diseñador seguir estos dos principios:

- A una interfaz de programación no una aplicación para que un objeto del cliente no tiene que ser consistente del objeto real que se utiliza al servicio de una determinada solicitud.
- Beneficiar la composición de objetos en la herencia de clases, lo cual no es negar la utilidad de la herencia como un mecanismo de reutilización, no se debe utilizar en exceso.

Marco de plantilla

- Un marco de plantilla es un conjunto de clases que constituyen una aplicación incompleta y genérica. Se establecen dos tipos de marcos:

- Marco de caja blanca: conjunto de clases denominadas abstracciones, están definidas tanto la interfaz como la implementación. Para especializarlo hay que implementar subclases de esas clases.

- Marco de caja negra: tiene definidas unas interfaces denominadas papeles. Para especializarlo hay que añadir clases que implementen sus papeles.

Nombre	Ventajas	Inconvenientes
Marcos de Plantilla	Reducen el trabajo de programación y mantenimiento. Minimizan la codificación y la puesta a punto, ya que proporcionan subsistemas que funcionan. Proporcionan arquitectura para el software. Desarrollan pequeñas aplicaciones que encajan dentro de los marcos, en lugar de las aplicaciones monolíticas. Son una base para la industria de componentes del software	Limitan la flexibilidad. Dificultad de aprendizaje. Reduce el grado de creatividad del trabajo de los desarrolladores

Tabla 166.Ventajas e inconvenientes, marco de plantilla

Comparación entre marcos y patrones

Los patrones son menores, los marcos de plantilla contienen varios patrones
Los patrones son más abstractos. La aplicación de un mismo patrón produce resultados diferentes.

Los patrones son menos especializados. Ya que se puede emplear en distintos tipos de aplicación

Relaciones de diseño

Proxy

GoF clasifica este patrón como objetos estructurales, mientras que en el esquema utilizado en Buschmann, Meunier, Rohnert, Sommerland (BMRSS), se clasifica como de control de acceso. Se puede utilizar para describir de la siguiente manera mediante plantilla.

- Nombre: Proxy.

- También conocido: Sustituto.

- Problema: El servidor proxy aborda el problema en la provisión directa de acceso a un objeto real que representa una carga significativa como es: el tiempo de carga, el espacio, la comunicación, etc. Como tal, el uso de proxy puede ser particularmente relevante en el desarrollo de sistemas distribuidos.

- Solución: Se trata de proporcionar un representante del objeto, y permitir que el objeto de usuario pueda comunicarse con el proxy en lugar de lo real. El proxy proporciona la misma interfaz que el objeto real y garantiza el correcto acceso a este objeto, mientras que, posiblemente, también llevar a cabo tareas adicionales tales como hacer cumplir las normas de protección de acceso. En caso necesario, puede organizar la creación y supresión de las instancias del objeto real.

- Aplicación: Proxy se refiere al acceso de los objetos relativamente complejos que pueden estar en un espacio diferente, como una base de datos remota (proxy remoto), implican gastos significativos en la creación de objetos.

- Estructura: Muestra este papel en términos de diagrama de clases y diagrama de secuencia que describe una posible forma de trabajar.

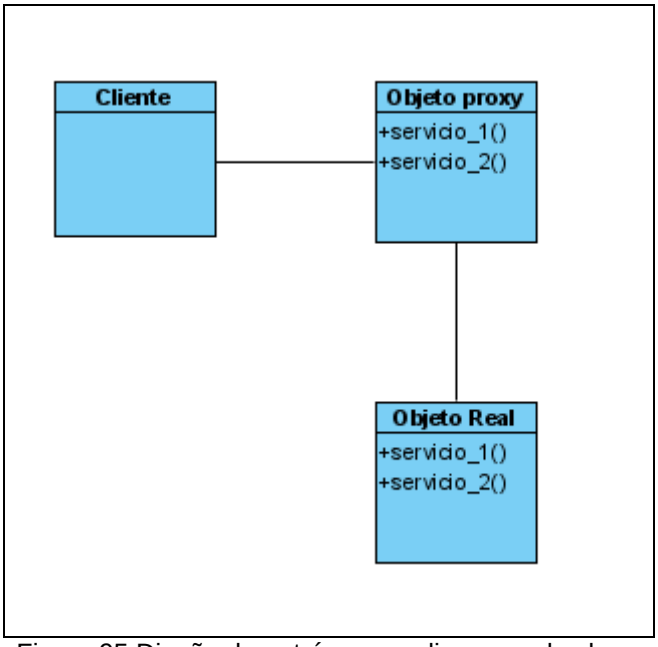


Figura 95.Diseño de patrón proxy diagrama de clases

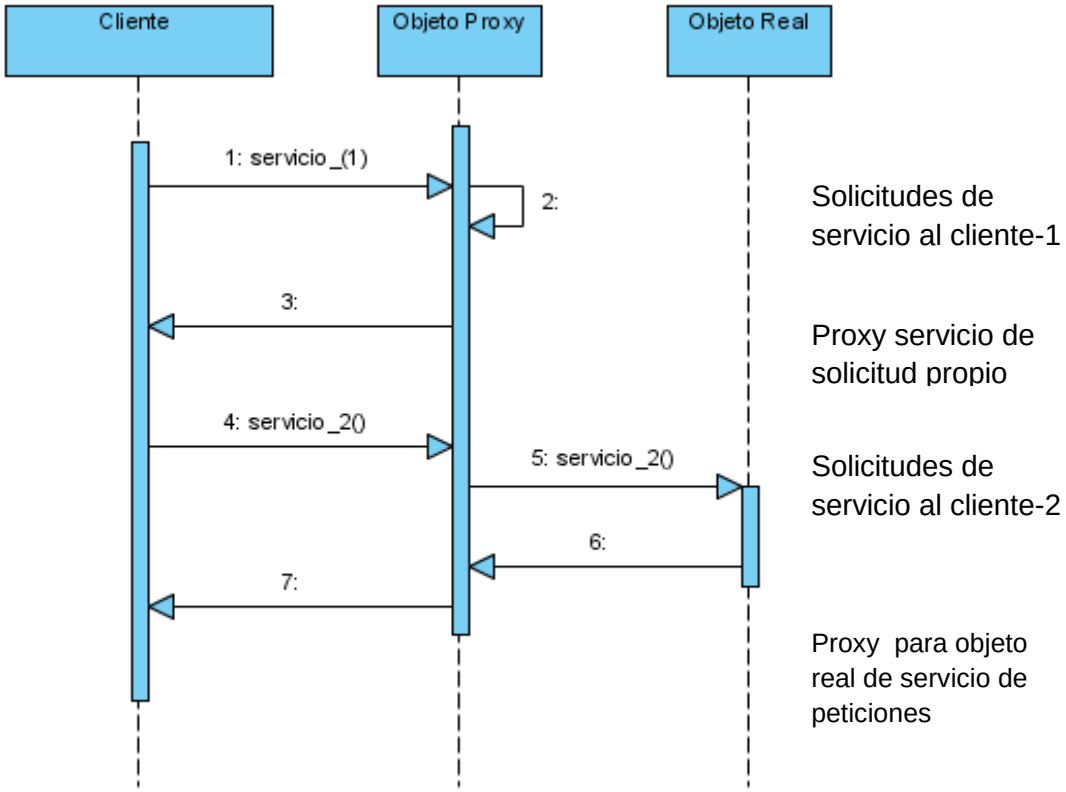


Figura 96.Diseño de patrón proxy (diagrama de secuencia)

Implementación: Tiene las siguientes características:

- Admite la sobrecarga del operador de acceso a miembros. El proxy se comporta de igual manera que un puntero. Sobrecargar el operador permite desarrollar tareas que se referencian a un objeto.

- Smalltalk proporciona un enganche que se puede utilizar para permitir el reenvío automático de peticiones
- Los proxies no tienen porqué conocer el tipo del sujeto real.

Usos conocidos: Tanto GoF y BMRSS son muy extensos y variados porque están basadas en aplicaciones web donde este patrón es muy utilizado.

Relacionados de patrones: Relacionados con el adaptador y Decorador, los patrones están preocupados por los problemas de interconexión.

Consecuencias: El proxy puede ofrecer pocos beneficios e incluso imponer una sobrecarga leve. Proxy también agrega un nivel de indirección de acceso a objetos, que es una ventaja de diseño (en su mayoría), pero puede añadir gastos generales en cuanto a su implementación.

Diseño de los atributos: Los nombres de los tipos incluyen el nombre del patrón, es una ventaja de poder comunicar a los que leen el código o los diagramas de diseño que se va a usar.

4.3.7.7.3 Ejemplo idiomas

UML diagrama de estructura compuesta y una combinación de los diagramas de clases UML y el diagrama de paquetes UML

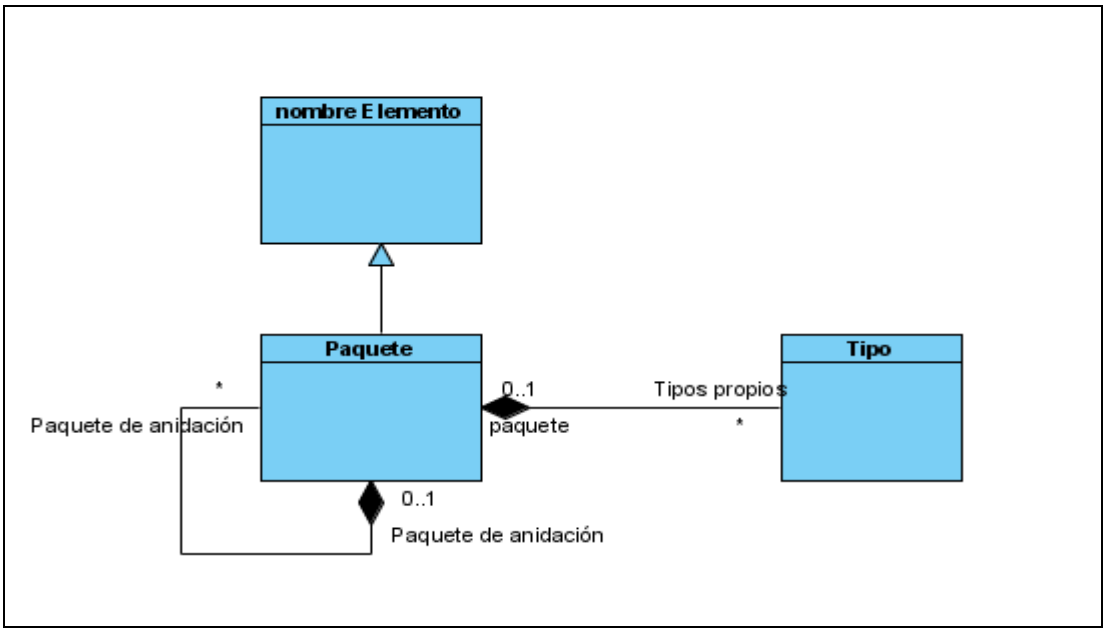


Figura 97.Las clases se definen en el diagrama de paquetes fuente UML

- Nombre Elemento:

    - Descripción: representa elementos con nombre.
    - Generalización: Elemento.
    - Un elemento es un componente de un modelo.
    - Semántica.

Los elementos con nombres son instancias de elemento con nombre. El nombre de un elemento con nombre es opcional. Si se especifica, entonces es válida cualquier cadena, incluyendo la cadena vacía que se puede utilizar.

- Paquete

  - Descripción: Un paquete es un contenedor de tipos y otros paquetes.
  - Generalizaciones: Nombre elemento.
  - Atributos:
    - Paquetes anidados: Paquete [*]
      El conjunto de paquetes de contenidos.
    - Paquetes anidados: Paquete [0 .. 1]
      El contenido del paquete.
    - Tipos propios: Tipo [*]
      El conjunto de tipos de contenidos.

- Tipo

  - Descripción.
  - Un tipo puede estar contenido en un paquete.
  - Generalizaciones.
  - Nombre elemento.
  - Atributos:
    - Paquete: El paquete [0 .. 1]
      El contenido del paquete.

4.3.7.8 Punto de vista de la interfaz

El punto de vista de la interfaz de la información ofrece a los diseñadores, programadores y probadores los medios para saber cómo utilizar correctamente los servicios prestados por un tema de diseño.

NOTA: Las interfaces de usuario no las he contemplado ya que se tratan por separado.

Cuando analizamos un sistema creamos un modelo que representa un aspecto de la realidad que nos interesa. Este análisis lo realizamos mediante la metodología orientada a objetos, aquí modelamos el mundo en términos de tipos de objetos y como interaccionan entre ellos. Este modelo se convierte en un diseño orientado a objetos que más adelante se implementa con la programación orientada a objetos.

Para el estudio de las interfaces mediante el modelo de objetos hay muchos métodos que se pueden realizar desde este punto de vista como son:

- Método jerarquía de diseño orientado a objetos (HOOD)

  Este método de diseño con respecto a su integridad se puede resumir de la siguiente manera:

  - El modelo de diseño ideal es que captura los puntos de vista: comportamiento, funcionalidad, construcción y modelado de datos, de manera equilibrada. Sin embargo, el énfasis en el análisis textual en la construcción es una descripción que contiene una mezcla de los aspectos del comportamiento, funcional y modelado de datos, y el proceso de extraer el punto de vista de construcción del modelo a partir de esto está lejos de estar bien estructurado.

  - La representación esquemática no es muy utilizada en el proceso de diseño, y no ayuda a los procedimientos de diseño de cualquier manera.

  Desde el punto de vista de construcción HOOD por su forma esquemática en realidad es solo adecuada para describir el resultado final del proceso de diseño. Así que el método en sí sigue siendo en gran medida basado en la manipulación de las descripciones textuales del modelo de diseño, esto es inadecuado para su uso con sistemas grandes.

- Método de Fusión

  El Método de fusión, representa un avance muy sustancial. Es capaz de abordar problemas del "objeto" y ofrece un proceso con el apoyo de un conjunto de técnicas de análisis y diseño. Sin embargo, es un requisito de la fusión que las decisiones sobre la elección de los objetos se hagan en una fase temprana. Esto es crítico en la forma del diseño final.

- Método Proceso unificado

  Sus estructuras de uso se apoyan en:

  - Abstracción: Tanto en términos de tipos de datos y a través del mecanismo genérico, con este último permite una abstracción en parte de la descripción de los algoritmos
  - Encapsulación: El alcance de los tipos de los datos puede ser controlado a través del mecanismo público/privado.
  - Modularidad, a través de la utilización del paquete de construcción (y podría decirse que mediante la tarea y los mecanismos de procedimiento).

  Un problema con este método es la evolución en la complejidad del paradigma orientado a objetos en el sentido de que los métodos o en parte, tienden a avanzar a través de un proceso de agregación, con

consecuencias obvias en términos de la complejidad de los procesos y la variedad de notaciones.

- Marcos o estructuras de orientación a objetos

  El marco en la aplicación orientada a objetos ofrece otro mecanismo de reutilización, pero se basa en la reutilización de objetos físicos. Es menos abstracto que el patrón de diseño (que se puede utilizar para describir los marcos). Los marcos están más relacionados con la aplicación del diseño y establece el concepto de biblioteca. Los marcos proporcionan:

  - Las funciones del sistema.
  - Sistema de comunicación.
  - Comportamiento del sistema.
  - Descomposición conceptual.
  - Funciones de los componentes.
  - Comportamiento de los componentes.
  - Componente de la comunicación.

De todos los métodos proporcionados anteriormente he elegido el método de los marcos orientado a objeto y para explicar el funcionamiento de las interfaces he elegido el software por componentes CORBA que lo explico con más detalle en el apartado siguiente.

4.3.7.8.1 Consideraciones de diseño

Una descripción de la interfaz funciona como un contrato vinculante entre los diseñadores, programadores, clientes y evaluadores.

El marco en la aplicación orientada a objetos ofrece un mecanismo de reutilización, pero se basa en la reutilización de físico de los objetos. Algunas de las principales características del concepto de marco orientado a objetos se definen a continuación:

- Reutilización a través de la modularidad. En vez de usar jerarquía como base de reutilización, los marcos orientados a objetos utilizan el concepto separación de las preocupaciones a través de la modularidad es decir se identifican abstracciones, encapsulación, modularidad y la jerarquía como los elementos primordiales del modelado de objetos. Realizamos hincapié, “el mecanismo de intercambio de datos permite la transferencia de datos entre: hombre–programa, componentes y recursos del sistema”.

- Definición de los componentes genéricos: En un nivel superior es una ampliación del concepto de biblioteca. En el caso de subprogramas biblioteca para obtener éxito, la identificación de las operaciones funcionales tienen que estar definidas y que se puedan parametrizar y

por consiguiente volver a usar con facilidad. Estos criterios se pueden emplear para los marcos aunque para lograr el éxito en la parametrización en la unidad del sistema es un objetivo mucho más complejo

- Inversión de control: Tiempo de ejecución de carácter arquitectónico permite la personalización de la gestión de eventos dentro de un marco. Cuando un evento tiene lugar, el distribuidor del marco realiza el evento invoca la solicitud para realizar operaciones especificas de la aplicación y luego permite que el marco en lugar de cada aplicación determine un conjunto de métodos específicos de la aplicación para dar respuesta a eventos externos.

Este énfasis en la reutilización de código actual implica que el concepto de marco como el de biblioteca puede ser eficaz cuando los elementos de un sistema realizan tareas necesarias para una variedad de aplicaciones diferentes. Los marcos han tendido a ser útiles en la realización de funciones bien definidas como son: interfaz gráfica de usuario y middleware.

El middleware: comunicaciones que proporcionan una capa de operaciones de enlace potencialmente, los objetos a distancia a través de tales mecanismos como el Object Request Broker (ORB), con CORBA siendo probablemente uno de los ejemplos más conocidos de esta forma.

Fayad y Schmidt clasifican los marcos como:

- Marcos de caja blanca: utilizan mecanismo orientado a objetos como la herencia y el polimorfismo como medio de configuración. Ya que permite la estructura visible la clase padre y tienden a ser específicos del lenguaje en su forma. Como ejemplo de marco de caja blanca la estructura de java Applet. La clase abstracta Applet define operaciones que pueden realizar applets.

- Marcos de caja negra: Define interfaces que permiten a otros elementos ser conectados en el código del marco. No es necesario que un usuario conozca los detalles de funcionamiento del marco, la realización de los marcos se hace más difícil ya que el diseñador marco debe prever todas las formas en que el marco se utilizará. También hay más margen para hacer un marco independiente del lenguaje de programación específico como ocurre con CORBA.

Para ello vamos a realizar un estudio para un software de componentes como es CORBA en el cual se especifica con detalle el concepto de interfaz: CORBA es una arquitectura para sistemas de objetos distribuidos desarrollado por la OMG.

Establece especificaciones de un software intermedio orientado a objetos diseñado para ofrecer portabilidad dentro de una red de sistemas heterogéneos. OMG es la misma asociación que desarrolló UML y ha publicado

una arquitectura común de solicitud de objetos. Los conceptos más influyentes para la descripción de CORBA:

1. Interfaz: En el software orientado a objetos, los objetos se comunican por medio de mensajes en los que se solicitan operaciones y valores de los atributos. Se distingue entre una clase y la interfaz(ces) que implementa. Por ende en el caso de software no distribuido el objeto cliente y el objeto servidor están en el mismo proceso y la clase y la interfaz que la implemente también.

   Un software distribuido: El objeto cliente y el objeto servidor pueden estar en diferentes procesos. Y por tanto en el proceso del cliente no está la clase del servidor sino solo su interfaz. De este modo CORBA solo considera interfaces ya que el concepto de clase no se requiere de la comunicación entre cliente y el servidor. Los objetos implementan las interfaces ya que cada operación se solicita a un objeto específico.

2. Objeto: Es una entidad que proporciona servicios a las entidades que lo solicitan. Un objeto posee identidad, interfaz e implementación.

3. Referencia a un objeto: Es algo que identifica el mismo objeto cada vez que se usa.

4. Tipo de objeto: Un tipo de objeto corresponde con una interfaz ya que los objetos referenciados de un tipo satisfacen la interfaz.

5. Petición: Es el mensaje de un objeto cliente a un objeto servidor para pedir la ejecución de un método. Una petición debe contener: identificación del objeto, operaciones que pueden tener o no parámetros y estos parámetros son idénticos a los que usa UML.

   La petición se puede invocar de dos maneras:

   - Estática: se determina en tiempo de compilación la interfaz que se usa
   - Dinámica: se determina en tiempo de ejecución es decir la interfaz se debe haber compilado antes.

6. Operación y método.

7. Objeto intermediario de peticiones (ORB): Es responsable de buscar el objeto servidor dentro de la red. Proporciona una variedad de servicios que hacen viable que los componentes reutilizables se relacionen con otros componentes independientes de la ubicación dentro del sistema.

Lenguaje de definición de interfaz (IDL)

IDL no es un lenguaje de programación, ya que no genera código objeto o ejecutable a partir de sí mismo. Las implementaciones de CORBA pueden generar código fuente principalmente en C++, Java, Smalltalk, COBOL a partir de IDL. La interfaz definida con este lenguaje genera en tiempo de compilación las siguientes salidas:

- Stud.
- Un esqueleto.
- Un archivo de cabecera, incluye tipos de datos como estructuras y constantes y aplicaciones de los clientes y servidores.

4.3.7.8.2 Elementos de diseño

Descripción de la arquitectura CORBA

- Aplicaciones del cliente: Los clientes efectúan solicitudes mediante peticiones de operaciones a los objetos de los servidores que son realizadas mediante invocación ya sea estática o dinámica.

- Stub: Establece correspondencia mediante las operaciones de la interfaz y los hábitos que dependen del lenguaje de programación que invoca a la aplicación del cliente cuando hace una solicitud.

  Se producen mediante el lenguaje de definición de interfaz y se conecta con la aplicación. Los lenguajes de programación orientada a objetos no necesitan stubs.

- ORB al servidor: Obtiene las solicitudes de ejecución de módulos. Se conecta con la aplicación del servidor. Da formato a los parámetros e invoca los métodos mediante su ejecución a partir del esqueleto.

- Depósito de interfaces: Contiene las interfaces, constantes y definiciones de los tipos que usan.

- ORB al servidor: obtiene las solicitudes de ejecución de módulos. Se conecta con la aplicación del servidor. Da formato a los parámetros e invoca a los métodos mediante su ejecución a partir del esqueleto.

- Adaptador de objetos portables (POA): Llevan tareas relativas a la implementación de objetos. Los adaptadores de objetos se unen con la aplicación. Puede haber varios POA que formen una jerarquía para un ORB, cada uno de ellos gestiona un grupo de

objetos. Aplicaciones al servidor: Incluyen una implementación de los objetos y métodos, código de inicio y fin de la aplicación.

- Métodos: Cada método es el código que implementa un objeto para una determinada operación de la interfaz. El conjunto de las implementaciones de las operaciones de la interfaz y el conjunto de los métodos se denomina “sirviente”. La implementación de los objetos debe ser independiente del ORB.

- Servicio de los objetos y servicios comunes: Se detallan en el apartado 4.3.7.8.2.1.

- Depósito de las implementaciones: Localiza las interfaces solicitadas dentro del depósito de las interfaces y contiene información que permite al ORB localizar y activar las implementaciones de un objeto al que se ha hecho referencia.

4.3.7.8.2.1 Atributo de interfaz

Una descripción de cómo otras entidades que interactúan con esta entidad. El atributo de interfaz describe los métodos de interacción y las normas que regulan estas interacciones.

Servicios de los objetos

Presentan los siguientes canales de servicio:

1. Características comunes

   La manera en que se usan los objetos en las especificaciones de los servicios, aunque no forma parte del entorno CORBA en una manera estricta. Los objetos presentan las siguientes características:

   - Se usan las interfaces para unificar los objetos.
   - Separación entre interfaz e implementación.
   - Uso de la herencia incluso múltiple entre interfaces para ampliar la funcionalidad y permitir su evolución.

   Los servicios en los objetos presentan los siguientes principios.

   - Las interfaces presentan implementaciones con diferentes calidades de servicio
   - Las excepciones permiten comunicar situaciones extrañas
   - Los objetos CORBA están implementados de manera que se pueden activar local o lejanamente.

2. Servicios de nombre

   Un objeto puede poseer varios nombres. Un contexto de nombres es un espacio de nombres de los objetos en el cual estos nombres no deben ser iguales. El servicio de nombres proporciona una organización de nombres de objetos que sirve para identificarlos mediante otros objetos.

3. Servicios de eventos

   Para CORBA, un evento mantiene relación con un objeto y tiene interés por otros objetos; se hace accesible mediante una notificación que es un mensaje. El servicio de evento distingue tres tipos de objetos:

   - Suministradores: Generan eventos.
   - Consumidores: Lo procesan.
   - Canales de eventos: Son suministradores y consumidores al mismo tiempo.

4. Servicio de notificaciones

   Es una ampliación del servicio de evento mediante la siguiente información:

   - Transferencia de eventos con estructura de datos complejas.
   - Posibilidad de añadir filtros a los proxies para que los clientes puedan especificar qué eventos quieren recibir.
   - Posibilidad de encontrar diferentes canales de servicio.

5. Servicio de relaciones

   El concepto de relación es semejante al de asociación en UML. El servicio de relaciones permite establecer relaciones: temporales y permanentes entre dos o más objetos sin modificarlos. Hay tres niveles:

   - Nivel base: Define relaciones y roles.
   - Nivel de grafo: Define objetos nodos y objetos de recorrido de un grafo, permite recorrer un grafo sin activar los objetos.
   - Presenta dos tipos de relación: Por contenido y referencia.

6. Servicio de ciclo de vida

   Proporciona operaciones para crear, copiar, mover, y borrar objetos de forma local o lejana. Soporta condiciones de integridad referencial entre objetos y asociaciones entre grupos de objetos.

7. Servicio de propiedades

   Sirve para definir atributos de los objetos dinámicamente, en cambio IDL lo hacía de manera estática. Los atributos dinámicos poseen un nombre,

tipo y valor en el cual se pueden leer y modificar. El servicio de propiedad puede tomar los siguientes valores:

- Normal: Sin restricciones.
- Solo lectura: Permite leer y borrar.
- Fijo-normal: Permite modificar pero no borrar.
- Solo lectura fija: Permite sólo leer.

8. Servicio de Tiempo

   Se utiliza para obtener la hora, corroborar el orden en que se han realizado los distintos eventos. El servicio de tiempo incorpora los siguientes servicios:

   - Servicio básico de la hora: Abarca operaciones para conseguir y manipular la hora
   - Servicio de eventos de temporización: Otorga operaciones de gestores de eventos por tiempo y gestiona los eventos que producen.

9. Servicio de externalización

   Significa transformar en un objeto flujo. El servicio de externalización permite realizar lo siguiente:

   - Objeto flujo: Área de datos con un cursor está en memoria, disco, envía a la red.
   - Sirve para facilitar la exportación/importación de un objeto: proceso, ordenador, ORB, se realizará una internalización para establecer un objeto nuevo.

10. Servicio de estado persistente

    Guarda el estado de un objeto en memoria permanente y recuperarlo. Presenta los siguientes componentes:

    - Objetos de almacenamientos: Presenta un tipo en el cual hay asociados atributos, operaciones de estado y herencia si las hubiera.
    - Almacenes de datos: Se compone de almacenamiento de origen posee un tipo que consta del objeto de tipo de almacenamiento que puede tener y operaciones. En un almacén de datos cada almacenamiento de origen transmite no solo sus objetos sino todos los tipos de familia que encabeza.

11. Servicio de seguridad

    Presenta los siguientes aspectos:

- Confidencialidad: Se permite el acceso a la información a usuarios autorizados.
- Integridad: Los usuarios autorizados pueden modificar la información y de manera autorizada.
- Responsabilidad: Los usuarios deben ser responsables de sus acciones con respecto a la seguridad.
- Disponibilidad: El acceso al sistema no se puede negar al usuario injustificadamente teniendo en cuenta los siguientes aspectos:

    - Identificación: Diga quién es, es decir es una persona u objeto titular para la autorización de acceso.
    - Autenticación: Es quien dice que es. La persona u objeto titular
    - Autorización: De una persona u objeto titular
    - Control de acceso: Permitir el acceso previamente identificado y autenticado.

- Auditoría de Seguridad: Determinar a los principales responsables de las acciones en materia de seguridad e identificarlos mediante peticiones.
- Seguridad de la comunicación entre objetos: Requiere la autenticación del cliente y servidor a la vez.
- No repudio: Proporcionar pruebas irrefutables del origen de los datos y del recibimiento del destinatario.
- Gestión de políticas de seguridad.

12. Servicio de Biblioteca

    Son tipos distintos de agrupación de objetos, los elementos de una biblioteca son del mismo tipo o tienen la misma interfaz. Los distintos tipos de biblioteca difieren en el orden de los objetos, el acceso, objetos por clave, criterio de igualdad de los objetos o puede haber objetos clave con el mismo valor.

    Distintas combinaciones de estas restricciones dan lugar a diferentes tipos de biblioteca. Por lo tanto a cada tipo de biblioteca le corresponde una interfaz.

13. Servicio de consultas

    Las consultas no solo se limitan a accesos de lectura sino que son instrucciones con predicados declarativas y pueden comprender valores de atributos e invocaciones y otros servicios de objetos. La solución de una consulta puede ser una biblioteca que se alcanza por medio de objetos de una biblioteca fuente que un predicado establecido, o puede provenir de un evaluador de consultas que se examinaría a través de una colección virtual. El servicio de consulta tiene las siguientes características:

- Facilita operaciones: Selección, insertar, actualizar y borrar elementos que se encuentran en la biblioteca.
- Elementos afectados pueden ser: objetos persistentes, transitorios, locales, lejanos.
- En los predicados se pueden usar: atributos, operaciones, herencia y navegación mediante relaciones. Lo mencionado anteriormente se realiza por medio de interfaces de los elementos de los objetos.

14. Servicios comunes

    Son servicios que pueden ser compartidos por las aplicaciones. Se dividen en dos grupos:

    - Servicios horizontales: Usados en la mayoría de las aplicaciones.
    - Servicios verticales: Son específicos de un dominio.

4.3.7.8.3 Ejemplo idiomas

Tomaremos como ejemplo el diagrama UML de componentes explicado anteriormente en la sección 4.3.7.3.

4.3.7.9 Punto de vista de interacción

El punto de vista de interacción define las estrategias para la interacción entre las entidades, en cuanto a por qué, dónde, cómo y en qué nivel se producen las acciones.

punto de vista del diagrama de secuencia

Proporciona una descripción del comportamiento de un sistema. Se trata de una notación de la caja blanca, porque su principal preocupación es describir cómo una respuesta de comportamiento en particular se va a producir. Al igual que muchas formas de comportamiento que son intrínsecamente no jerárquicos y, de hecho, un problema que esto puede crear fácilmente es la dificultad de comprender diagramas relativamente grandes y complejos.

Los diagramas de secuencia se refieren esencialmente a modelar la secuencia detallada de las medidas adoptadas por un conjunto de elementos de colaboración (que puede considerarse simplemente como hilos de procesamiento para este propósito), y con los mensajes que utilizan para coordinar estas acciones. Esto es obviamente un papel importante al ampliar los modelos de diseño orientados a objetos.

Otra área donde el diagrama de secuencia puede ofrecer una forma útil de visualización es en la descripción de los protocolos. En muchos sentidos, las interacciones de los objetos pueden ser considerados como una forma de protocolo, pero suelen emplearse más en términos para la descripción de las secuencias que se utilizan en redes de comunicaciones.

La principal limitación es la falta de una organización jerárquica, aunque a veces puede haber un margen de estructura a través de subgrupos de secuencias. Como regla general, las descripciones de las secuencias que abarcan más de una sola página pueden ser difíciles de comprender y manejar.

Punto de vista del diagrama de colaboración

El diagrama de colaboración hace un uso limitado de las notaciones de caja y la línea para el diseño detallado, y la principal forma en que se emplea es el de la gráfica de interacción de objetos. Lo interesante es que este diagrama de colaboración se construye para cada operación del sistema aunque, de nuevo, esto no es incompatible con la arquitectura orientada a objetos, que tiende a poner menos énfasis en cuestiones generales de construcción.

La notación de base cuenta con cuadros para representar objetos de diseño, unidos por flechas etiquetadas que representan el paso de mensajes. Una caja y sólo una, tiene una flecha externa como entrada, lo que representa el funcionamiento del sistema que el diagrama de colaboración tiene por objeto aplicar.

4.3.7.9.1 Consideraciones de diseño

Para los diseñadores. Esto incluye la evaluación de la asignación de responsabilidades en la colaboración, especialmente en la adaptación y la aplicación de patrones de diseño, el descubrimiento o la descripción de las interacciones en términos de mensajes entre los objetos afectados en el cumplimiento de las medidas necesarias.

Definiremos los siguientes conceptos:

1. Interacciones (desarrollo): Una interacción es la especificación del comportamiento de un caso de uso u operación en términos de secuencia de intercambio de mensajes entre objetos. Estos mensajes incluyen estímulos que pueden ser: peticiones de ejecución, operaciones, señales.

   Representan un comportamiento del sistema, presentan un hilo de ejecución que son secuencias de mensajes tal que el primero contiene un estímulo que provoca el envío del segundo y así sucesivamente es decir si un mensaje X ha sido causa para que se emitiera el mensaje Y entonces denominaremos al mensaje X es el predecesor de Y, y el mensaje Y es el sucesor del mensaje X.

2. Colaboraciones (contexto): Se tiene que indicar qué parte del modelo estático se usa. Una colaboración es un conjunto de papeles de clasificadores o instancias y de papeles de asociaciones que intervienen en una interacción o también se puede definir una colaboración en lo que respecta a clasificadores o por lo que respecta a instancias. Las instancias están enlazadas con otras instancias.

UML ofrece dos diagramas para representar una interacción y colaboración:

- Diagrama de colaboración: Pone énfasis en la descripción de la colaboración.
- Diagrama de secuencia: Pone énfasis en la sucesión temporal de los mensajes de la interacción.

Responsabilidades en los métodos

UML define una responsabilidad como la obligación de un clasificador. Las responsabilidades están relacionadas con las obligaciones de un objeto en cuanto a su comportamiento. Estas responsabilidades se basan en los siguientes tipos:

- Responsabilidades de hacer un objeto

    - Realizar algo el mismo: Crear un objeto, realizar algún cálculo.
    - Empezar un evento entre instancias (objetos).
    - Coordinar tareas en los objetos.

- Responsabilidades de conocer un objeto

    - Comprender los datos encapsulados es decir datos privados.
    - Comprender las instancias (objetos) que se relacionan.
    - Comprender las cosas que pueden derivarse o calcularse.

Cabe destacar que una responsabilidad no es lo mismo que un método, los métodos se implementan para llevar a cabo responsabilidades.

Las responsabilidades en los diagramas de interacción

Los diagramas de interacción forman parte del modelo de diseño del proceso unificado (UP). Los diagramas de interacción exponen opciones en la asignación de responsabilidades; esto se refleja en los mensajes que se envían a diferentes clases de objetos.

Estas opciones que se reflejan en los diagramas de interacción quedan expresadas en los patrones generales de software para asignar responsabilidades (GRASP) estos patrones describen los principios de diseño y la asignación de responsabilidades que se expresan mediante patrones de diseño.

Patrones de Principios Generales para Asignar Responsabilidades (GRASP)

Se definen los siguientes aspectos:

- En el diseño de objetos es primordial en la asignación de responsabilidades.
- La determinación de asignación de responsabilidades se crea durante los diagramas de interacción y con seguridad durante la programación.
- Un patrón es una solución a un patrón dado dentro de un contexto, con un nombre y principios relacionados con asignación de responsabilidades.
- Es muy importante comprender y aplicar estos aspectos durante la creación de los diagramas de interacción ya que para él diseñador su compresión constituye la base de cómo se diseñara el sistema.

Patrones GRASP	Descripción
Experto en Información	**Solución**: Clase que contiene la información imprescindible para desarrollar la responsabilidad. **Problema:** Durante el desarrollo de objetos, cuando se definen las interacciones de los objetos tomamos decisiones sobre la asignación de responsabilidades a las clases software. Esto conlleva que el sistema sea fácil de entender, mantener y extender, para la reutilización de componentes en aplicaciones futuras
Creador	**Solución:** para la creación se tienen que cumplir los siguientes acometidos. B añade objetos a A B incluye objetos de A B registra objetos de A B usa objetos de A B crea objetos de A **Problema:** La creación de objetos es una de las tareas en un sistema orientado a objetos. Por ello la importancia de contar con la asignación de las responsabilidades durante la creación. Ello conlleva a un soporte de bajo acoplamiento, mayor claridad, encapsulación y reutilización
Alta cohesión	**Solución:** Determinar responsabilidades para que la cohesión sea alta. **Problema:** Un elemento con responsabilidades altamente relacionadas y que no hace una gran cantidad de trabajo posee una alta cohesión. De las cuales pueden ser: clases, subsistemas, etc. Una clase con baja cohesión dificulta: Complicado de entender Complicado de reutilizar Complicado de mantener
Bajo acoplamiento	**Solución:** Determinar responsabilidades para que el acoplamiento sea bajo. **Problema:** Un elemento con bajo acoplamiento no depende de otros elementos, depende del contexto. De las cuales pueden ser: clases, subsistemas, etc.

	Una clase con alto acoplamiento dificulta: Son complicados de entender de manera aislada Son difíciles de reutilizar ya que dependen de la utilización de las clases de las que dependa
Controlador	**Solución:** Determinar responsabilidades de obtener o utilizar un mensaje de evento del sistema a una clase representa lo siguiente: Representa un escenario de caso de uso en el que tenga lugar el evento del sistema Representa el sistema global, dispositivo, subsistema. **Problema:** Un evento del sistema de entrada es un evento que es generado por una entidad externa en este caso el actor. Se agregan con operaciones del sistema como respuesta a los eventos del sistema, se relacionan mensajes y métodos

Tabla 167.Patrón GRASP

4.3.7.9.2 Elementos de diseño

Elementos de diseño de los diagramas de interacción:

Una instancia usa el mismo símbolo gráfico que el tipo es decir la cadena que lo designa subrayada. Su representación se asigna de la siguiente manera:

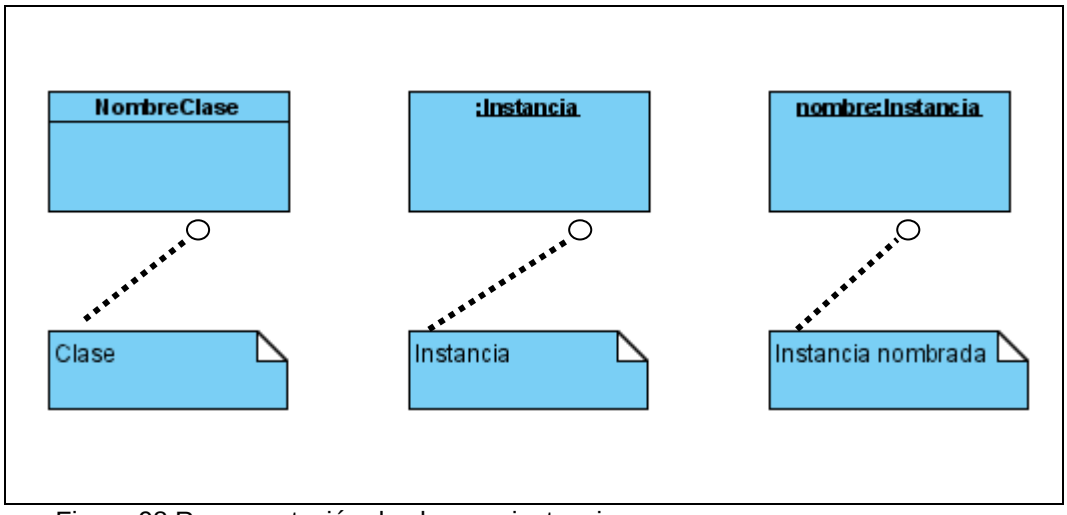


Figura 98.Representación de clases e instancias

En un diagrama de interacción para presentar una instancia de una clase se usa la representación de una clase, es decir un rectángulo con el nombre subrayado. También se puede usar para identificar la instancia de una manera específica o usar los ":" que preceden el nombre de la clase.

Sintaxis de la expresión de un mensaje

Valor de retorno:= mensaje (parámetro: Tipo de parámetro): Tipo de retorno.

Elementos que representan diagrama de colaboración

- Enlace

  Es la conexión entre dos objetos de la cual es posible la navegabilidad y visibilidad entre objetos, esto implica un enlace de asociación entre dos objetos. Durante un mismo enlace pueden fluir mensajes en ambas direcciones y múltiples mensajes.

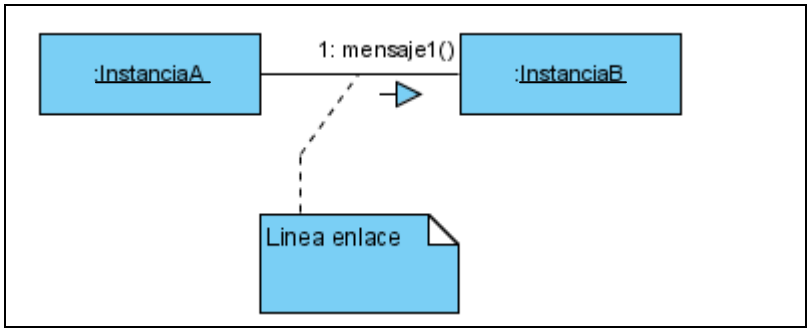


Figura 99.Representación línea de enlace

- Mensajes

  Representa la expresión de mensaje mediante dos objetos y una flecha que indica la dirección del mensaje. Pueden fluir varios mensajes durante el enlace. Se agrega un número de secuencia que indica el orden de cada mensaje en el hilo de control actual.

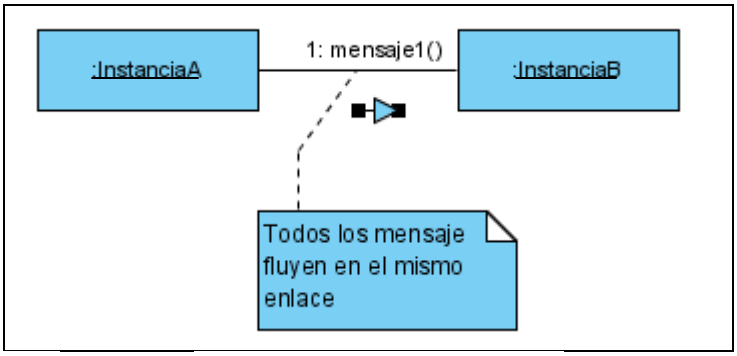


Figura 100.Representación de un mensaje

- Mensajes self o this: Representa un enlace hacia sí mismo, es decir mensajes que fluyen durante el enlace desde un objeto al mismo.

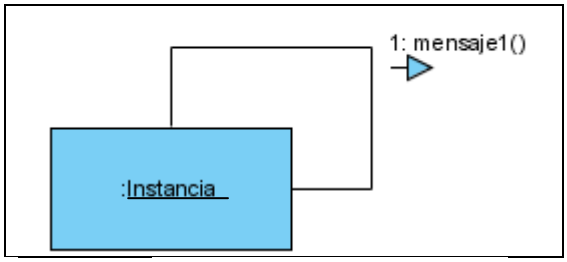


Figura 101.Representación mensaje 'this'

- Creación y destrucción de instancias

  Un mensaje se puede usar en la creación de una instancia se utiliza la palabra reservada create y para la destrucción de un mensaje se utiliza la palabra reservada destroyed y transient que equivale a las dos: create y destroyed y expresa que el objeto o enlace ha sido creado y destruido durante la interacción, el mensaje create podría incluir parámetros, indica el paso de valores iniciales. La propiedad de UML podría añadir "new" y "destroyed" a la caja de la instancia.

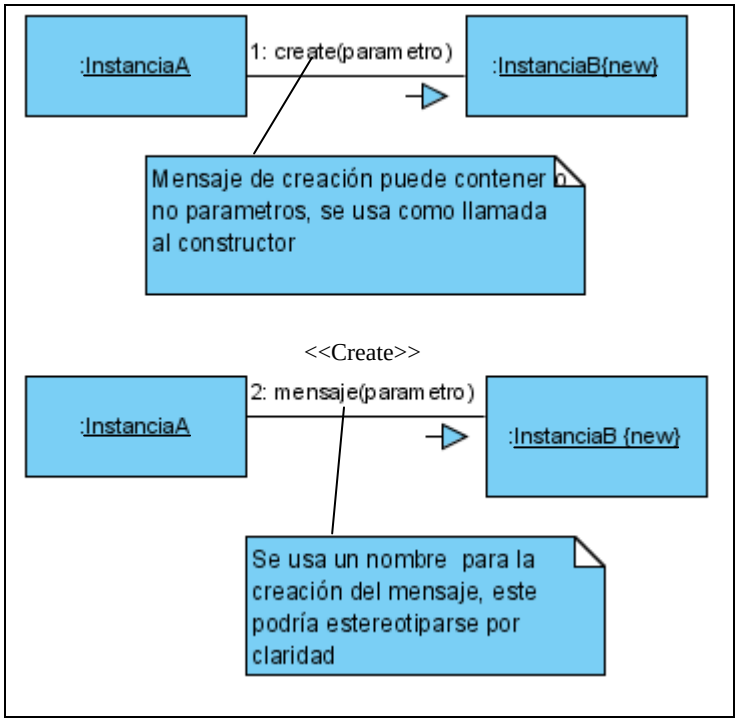


Figura 102.Creación de instancias

- Secuencia de números

  El número de secuencia de mensajes es consecutivo. El esquema de numeración es el siguiente:

  - En los diagramas de colaboración el primer mensaje puede enumerarse o no.
  - El orden y anidamiento de los siguientes mensajes se muestran con el esquema de numeración válido en que los mensajes anidados tienen un número adjunto. El anidamiento se denota

anteponiendo el número del mensaje entrante al número del mensaje saliente.

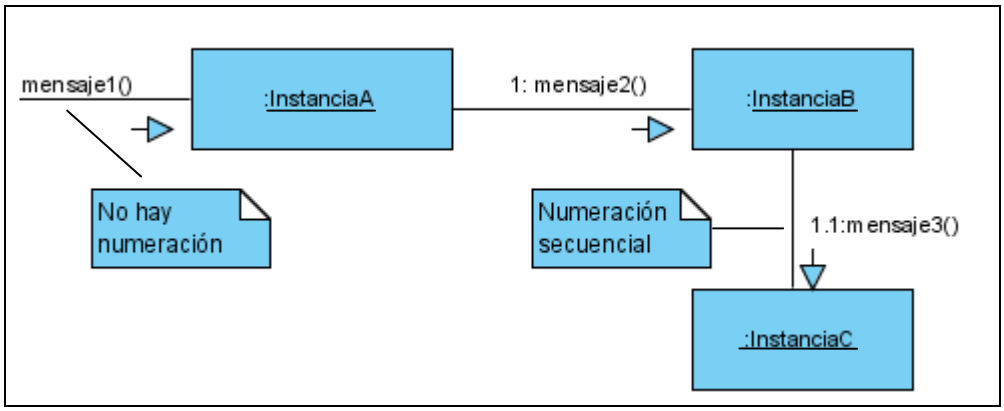


Figura 103.Representación secuencia de numeración

- Guarda

  Es la condición que se tiene que cumplir para que el mensaje sea enviado, además los mensajes predecesores que se hayan recibido.

- Expresión de secuencia

  Representa la siguiente sintaxis:

  Número de secuencia "["recurrencia "]" "," número de secuencia "["recurrencia "]"… ":".

  La recurrencia tiene el siguiente formato:

  "*" "["cláusula de iteración "]"
  Si la iteración es en paralelo en lugar del asterisco (*) solo aparece "*||", ó: "["cláusula de condición "]"

- La cláusula de condición

  Sirve para definir ramas de ejecución. Puede haber muchos números de secuencias con cláusulas de condición repetidas.

- Valores de retorno

  Especifica una lista de nombres con valores retornados si los hubiese. Presenta el siguiente formato:

  Valor de retorno "," valor de retorno… ":="

- Signatura

  Está compuesta por el nombre del estímulo y por una lista de argumentos entre paréntesis.

Elementos que representan diagrama de secuencia

El diagrama de secuencia a diferencia del diagrama de colaboración no se representa de forma explícita; los papeles de asociación quedan implícitos en los mensajes y se representa explícitamente el orden en el tiempo e incluso la duración de los mensajes y las operaciones que se ejecutan.

El diagrama de secuencia está estructurado en dos partes:

1. El tiempo se representa verticalmente de arriba abajo.
2. En dirección horizontal hay franjas verticales que corresponden a los diferentes papeles de clasificadores que participan en la interacción. Cada papel de clasificador está representado por el símbolo que encabeza su línea de vida. El orden de los clasificadores se lee de izquierda a derecha.

   - Línea de vida: Simboliza la existencia del papel en un cierto periodo de tiempo. Se representa mediante una línea discontinua vertical que va desde la creación hasta la destrucción.
   - Activación: Es una parte de la línea de vida durante la cual dicho papel ejecuta una determinada acción.
   - Enlaces: Los diagramas de secuencia no muestran enlaces.
   - Mensajes: Los mensajes entre objetos se representan con un mensaje sobre una línea con una punta de flecha entre los objetos que comienza en una activación. Su orden de arriba expresa el orden en que se producen en el tiempo.
   - Focos de control: Llamada de rutina ordinaria, muestra el periodo de tiempo en el cual un objeto se está ejecutando mediante una operación a la respuesta de un mensaje.

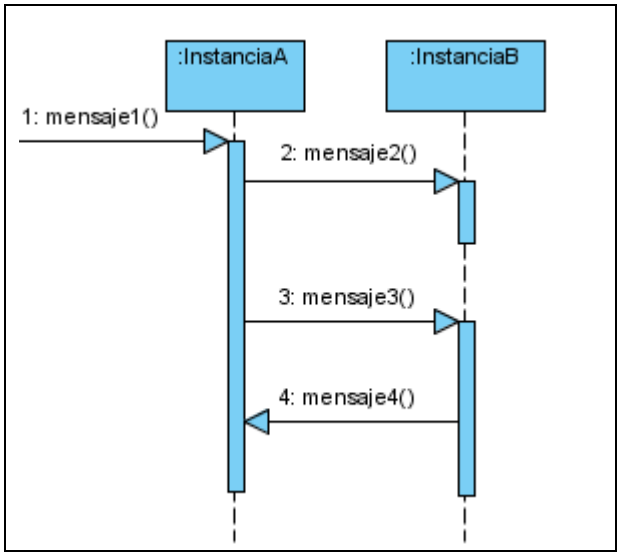


Figura 104.Representación de mensajes y focos de control

- Mensajes de retorno: Se puede mostrar el retorno de un mensaje al final de una activación en forma de flecha de línea discontinua y punta abierta.

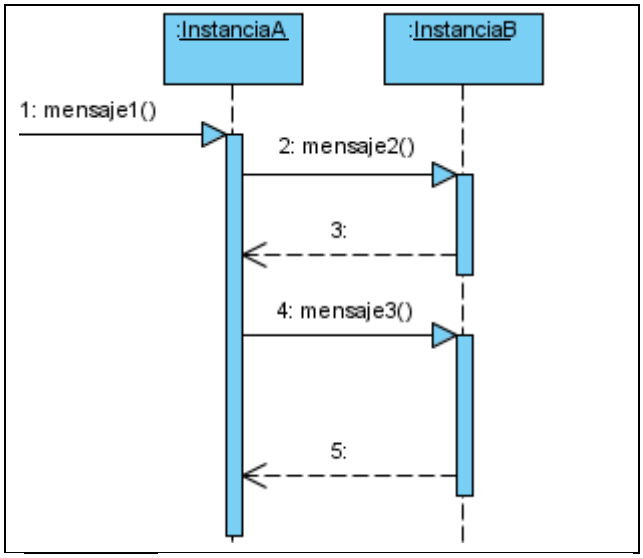


Figura 105.Representación de mensajes de retorno

- Mensajes Self o this: Representan un mensaje que se envía de un objeto a sí mismo utilizando una caja de activación anidada.

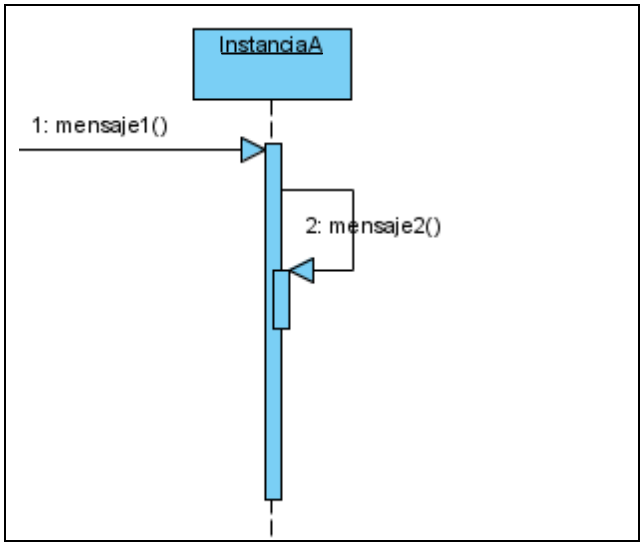


Figura 106.Representación mensajes this

- Creación de instancias: Los objetos creados se sitúan a la altura de la creación.

- Destrucción de objetos: Con la "X" y la línea de vida corta, indica la destrucción explicita del objeto.

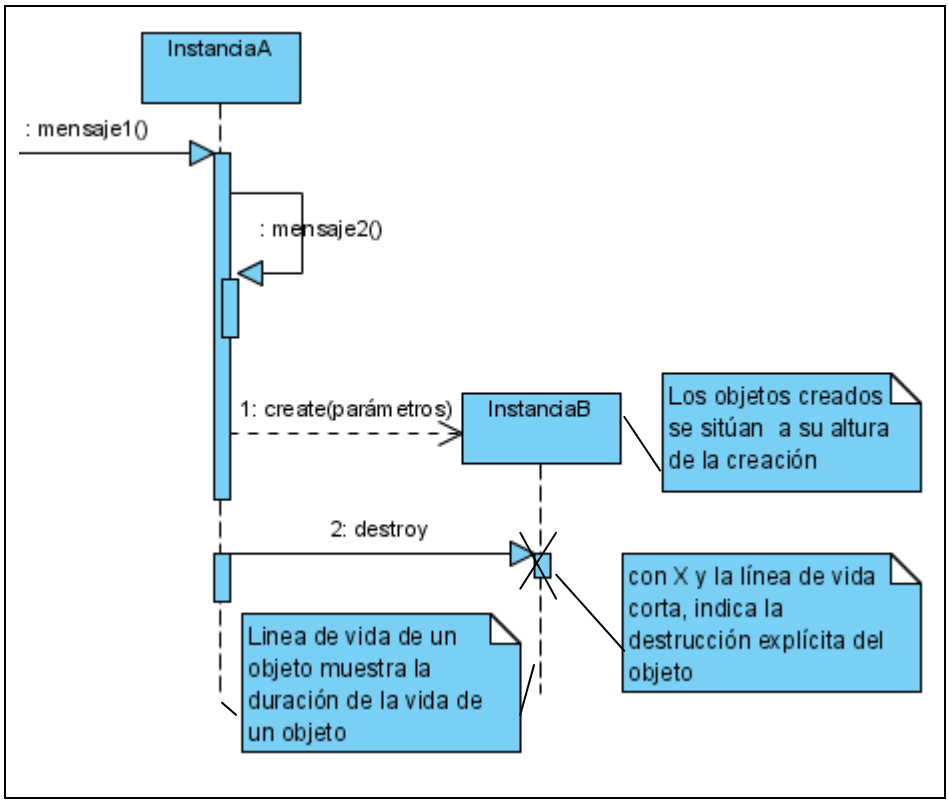


Figura 107.Creación de instancias y destrucción de objetos

4.3.7.10 Punto de vista del estado de la dinámica

El diagrama de estado se ocupa de proporcionar una descripción del comportamiento de un sistema. Hay objetos cuyo comportamiento puede variar a lo largo del tiempo, si esto sucede se dice que el objeto tiene estados. La información que contiene el diagrama de estado es redundante, es decir los cambios de estado son resultado de la dinámica de estado y por consiguiente de la ejecución de las operaciones de la misma clase o de otras, esto representa otro punto de vista de la dinámica de una parte del sistema que puede contribuir a su comprensión.

En el diagrama de estados se distinguen los siguientes elementos:

- Distintas maneras en que se puede encontrar un objeto, es decir los estados.
- Cambio de estado, es decir transiciones.
- Hecho que los produce, es decir acontecimientos.

Cabe resaltar que en UML se habla de máquinas de estado o de Harel; entre ellos existen diferencias. Nosotros utilizaremos diagrama de estados porque representa las máquinas de estado en forma de diagrama. El diagrama de estados se usa para describir objetos del dominio del usuario y se documenta por lo general en la etapa de análisis.

4.3.7.10.1 Consideraciones de diseño

Dinámica de sistemas, incluidas las modalidades, los estados, las transiciones, y las reacciones a los eventos.

Nombre	Descripción
Estado	Es una situación determinada dentro de la vida de un objeto o aquella en la cual se cumple alguna condición en la duración de una interacción, esto lleva implícita alguna acción o espera para que se produzca un evento. Un estado no corresponde a un instante en el tiempo, sino que el objeto o interacción permanece en un tiempo finito.
Transición simple	Consiste en que el objeto o interacción pasa de un estado origen a un estado destino. Esto sucede cuando se produce un evento y al mismo tiempo se cumple una condición específica, es decir una condición de guarda. Por lo tanto se ejecutan acciones y envío de mensajes a objetos o a un conjunto de objetos. Un estado puede tener transiciones de llegada.
Transiciones internas	Cabe destacar que una transición interna es distinta que una auto - transición, que es una transición ordinaria, el estado origen y el estado destino es el mismo. Las transiciones internas son seudotransacciones en las cuales no hay cambio de estado. Sirven para especificar acciones que se deben ejecutar en respuesta a un evento en el cual no provoca ningún cambio de estado en el

	objeto o interacción en cuestión.
Acción	Especificación de un proceso atómico, implica que no se ejecuta o se ejecuta hasta el final. Es posible describir una acción mediante un procedimiento o máquina de estados
Objetos por medio de mensajes	Pueden recibir solicitud de operaciones o señales. Las señales a diferencia de las operaciones no realizan ningún proceso, el único efecto es producir eventos.
Los eventos	Son hechos que cuando se producen pueden provocar transiciones de un estado a otro en objetos e interacciones o la ejecución de determinadas acciones. Un evento no está vinculado a ningún objeto o clase en particular. Un hecho puede afectar a los paquetes que contienen sus elementos en cuyo interior están definidos. Cada evento tiene un nombre que lo identifica dentro del paquete y puede tener parámetros.

Tabla 168.Conceptos básicos del diagrama de estados

Tipos de eventos

- Evento de llamada: Se produce cuando se llama una operación del objeto al que corresponde el diagrama. El mensaje de recepción es síncrono.

- Evento de señal: Representa la recepción de una señal por el objeto a la que corresponde el diagrama. El mensaje de recepción es asíncrono.

- Evento de cambio: Representa una notificación de que una condición ha llegado a ser cierta. Hay que hacer hincapié en que una condición de guarda se evalúa una vez se ha presentado el evento a una transición para determinar si se provoca la transición o no, mientras que esta condición produce el evento cuando ha llegado a ser cierta. Condición lógica: When(cuando).

- Evento de tiempo: Representa la notificación de que ha pasado un periodo de tiempo desde que se ha producido un evento o de que es una hora determinada. Ejemplo: se ha entrado en el estado origen de la transición: After(después)

Eventos internos

Son seudoacontecimientos, están vinculados a un estado en lugar de una transición sirven para poner en ejecución acciones que no van asociadas a ningún cambio de estado concreto.

Nombre	Descripción
Evento interno de entrada	Se producen cuando el objeto entra en el estado. No posee ni guarda parámetros, porque se nombran implícitamente cuando hay una transición de entrada en el estado, pero cuando no haya una transición interna, entonces no se producirá un cambio de estado. Posee la palabra entry, se especifican explícitamente.
Evento interno de salida	Son similares a la de la entrada, la diferencia que se producen a la salida del estado. Posee la palabra exit, se especifican explícitamente.
Evento interno de acción	Especifican acciones que se ejecutan cuando se llega al estado y acaban por sí mismas o cuando se sale del estado. Posee la palabra do, se especifican explícitamente.

Tabla 169.Tipos de eventos internos

Transiciones complejas

Un objeto o interacción puede estar en más de un estado al mismo tiempo. Una transición compleja con varios estados de origen solo tiene lugar si el objeto o interacción está en todos estos al mismo tiempo, se produce el evento a la transición es decir, se cumple la guarda si la hay y por lo tanto realiza una función de sincronización.

Una transición compleja se representa usando un seudoestado: es un símbolo que figura en una posición del diagrama donde habría un estado, no representa ningún concepto sólo tiene una función gráfica. Van a parar varias transiciones es decir, seudoestado de bifurcación, join pseudo-state o del que salen varias es decir, seudoestado de bifurcación, fork pseudostate; o ambas circunstancias al mismo tiempo.

Estados complejos

Un estado compuesto en el que hay varios subestados posibles, cada uno de los cuales puede ser un estado compuesto a su vez o no. A un estado compuesto le corresponde un diagrama de subestados. Los subestados de un estado pueden ser concurrentes es decir, que se presentan de forma simultánea o secuencial es decir, incompatibles entre sí. El diagrama de subestados consta de varias secuencias de subestado en paralelo. Cada secuencia comienza en un seudoestado inicial y acaba en un subestado final.

Las transiciones entre subestados de la misma secuencia se representan igual entre estados. Existen tipos de transiciones que tienen por estado de origen o de destino un objeto compuesto:

- Transiciones que entran a un subestado o salen del mismo.

- Transiciones que entran o salen del estado compuesto como una unidad. Si entra es como si la transición tuviera como estados de destino

los estados iniciales de todas las secuencias. Si sale es como si la transición tuviera como estados de origen los estados finales de todas las secuencias.

- Transiciones que tienen como destino un indicador de historia de los estados, el cual es un seudoestado que indica que en una transición de regreso al estado compuesto se vuelve dentro de este al mismo subsestado del que salió la última vez que partió del estado compuesto.

- Transiciones stubbed, estas transiciones tienen como estado destino u origen algún subestado de un estado compuesto del que no se especifica el diagrama de subestados.

Estados anidados

Un estado permite el anidamiento para contener subestados. Un subestado hereda la transición de su superestado.

Aplicaciones de los diagramas de estado

Un diagrama de estados podría aplicarse a una variedad de elementos de UML entre los cuales se hallan:

- Las clases.
- Los casos de uso.

Diagramas de estado de caso de uso

Una aplicación útil de los diagramas de estado es la descripción de la secuencia de eventos del sistema externo que reconoce y maneja un sistema en el contexto de un caso de uso. Los diagramas que describen los eventos del sistema global y sus secuencias en un caso de uso es una especie de diagrama de estados de casos de uso.

La utilidad de los diagramas de estado de casos de uso; el número de eventos del sistema y su orden para el caso de uso. Por ejemplo: procesador de texto resulta útil utilizar un diagrama de estados que ilustre el orden válido de los elementos externos.

Un diseñador puede realizar metódicamente un diseño que asegure el orden correcto de los eventos del sistema. En un dominio con varios eventos del sistema, la precisión y esmero de los diagramas de casos de uso ayudan al diseñador a asegurar que no se ha olvidado nada.

4.3.7.10.2 Elementos de diseño

Entidades de diseño:

Las transiciones se representan mediante flechas de punta coloreada que van del estado de salida al de llegada. Con la flecha se encuentra una expresión (denominada cadena de la transición) que presenta la siguiente sintaxis formal:

Signatura ` [´guarda`] `/`acción `^´ envío

Puede haber varias acciones o envíos o no haber ninguno, y puede estar mezclado. El orden en que aparecen es en el que se deben ejecutar.
A continuación veremos una explicación de cada uno de los elementos de la cadena de transición.

- Signatura: Tiene un formato que depende del tipo de evento. En el caso de un evento de llamada o de señal, se define así:

  Nombre evento `(` nombre parámetro `:` expresión tipo `,`….`)`
  Si el evento es el tiempo, la signatura adopta una de estas formas:
  `after (` expresión de tiempo´)`
  Donde expresión de tiempo consiste en una duración a partir de un origen, o:
  `when(´hora o fecha`)´
  Finalmente, si el evento es de cambio, tendremos lo siguiente:
  ´when(` expresión booleana´)´

  Nota terminología: Respecto al uso de términos parámetros y argumentos en relación con una llamada, hay que tener presente el parámetro correspondiente al punto de vista de los que se llame y argumento, al punto de vista de lo que hace la llamada.

- Guarda: Es una expresión que puede tomar el valor verdadero o falso, escrita en seudocódigo o en el lenguaje de programación que se utiliza.

- Acción: Es la especificación de una acción, en seudocódigo o en lenguaje de programación que se usa. En el caso de un evento diferido, debe aparecer la palabra clave defer como primera acción.

- Envío: Este elemento presenta la forma siguiente:
  Destino `. `Mensaje `(´ argumento `,`… `)`

  En que el destino tiene que identificar un objeto o grupo de objetos y el mensaje puede ser una operación del objeto de destino o una señal.
  Las señales se pueden definir como clases con el estereotipo signal. No tiene operaciones, y en el compartimiento de los atributos tienen los parámetros de la señal.

Pueden constituir jerarquías de herencia, pero no les es posible establecer relaciones de ningún tipo con clases. Las señales a las cuales deben ser sensibles los objetos de una clase se pueden especificar en un compartimiento adicional del símbolo de la clase

4.3.7.11 Punto de vista de algoritmo

La descripción del diseño detallado de las operaciones (tales como los métodos y funciones), los detalles y la lógica interna de cada entidad de diseño.

Seudocódigo

El seudocódigo puede ser utilizado tanto en el diseño arquitectónico como en el detallado; al igual que los diagramas de flujo, esto puede ser utilizado a cualquier nivel de abstracción; con el uso del seudocódigo, el diseñador describe las características del sistema usando frases cortas y concisas en español, las cuales se encuentran estructuradas por medio del uso de palabras clave como si – entonces – si no, mientras – repetir y fin. Con estas palabras y el uso del sangrado se puede describir el flujo de control del programa, mientras cada frase en español describe las acciones a ejecutar. Cada frase en español es expandida en un seudocódigo más detallado en el nivel inferior, hasta que la definición de las especificaciones llega al nivel del lenguaje de instrumentación. El seudocódigo puede reemplazar los diagramas de flujo y reducir una buena cantidad de documentación externa requerida para describir el sistema.

Diagramas de flujos estructurados

Los diagramas de flujo representan la forma más tradicional para especificar y documentar los detalles algorítmicos de un producto de programación, estos diagramas utilizan cajas rectangulares para especificar las acciones, cajas en forma de rombos para las proposiciones de decisión, arcos dirigidos para las interconexiones entre diversas cajas, así como una variedad de formas especiales para denotar las entradas, las salidas, los almacenamientos etc. las formas básicas se caracterizan por una entrada única y una salida única para cada una de las formas.

Estas formas pueden ser anidadas dentro de otras hasta el nivel deseado de anidamiento, manteniendo el principio de una sola entrada y una sola salida. Los diagramas de flujo estructurados son equivalentes al seudocódigo, se tiene el mismo poder de expresión; ambos pueden ser usados para expresar cualquier algoritmo posible. Los diagramas de flujo estructurado pueden ser preferidos en aquellas situaciones en donde la claridad de control deba ser recalcada. Los diagramas de flujo estructurados subrayan el flujo de los mecanismos de control debido a la naturaleza gráfica de su representación visual.

Lenguaje natural estructurado

El uso del lenguaje natural (dependiendo del idioma que interesa) en forma estructurada puede ser utilizado para proporcionar una especificación paso a paso de un algoritmo.

4.3.7.11.1 Consideraciones de diseño

El punto de vista algoritmo proporciona información que necesitan los programadores, los analistas de los algoritmos en cuanto a rendimiento en tiempo-espacio y la lógica de procesamiento antes de su aplicación, y para ayudar en la elaboración de planes de unidad de prueba.

El programador es el responsable de usar los recursos del computador de la forma más eficiente posible. Surge la necesidad de determinar cómo se ha de medir la eficacia de un algoritmo.

Una manera de medir la eficacia de un algoritmo es contar cuantas instrucciones de cada tipo se ejecutan, multiplicar ese número por el tiempo que emplea la instrucción en ejecutarse y realizar la suma para los diferentes tipos.

Ta= tiempo asignación de enteros
Tc= tiempo de una comparación de enteros
Ti= tiempo de incrementar un entero
Tv= tiempo de acceso a un elemento de un vector

Tabla 170.Tiempos que se utilizan en un algoritmo

La suma de todos estos tiempos da lugar a dos polinomios de la forma:

$$Tmin = An^2 – Bn + C$$
$$Tmax = A' n^2 – B'n + C'$$

Para los tiempos de ejecución mínimo y máximo, respectivamente, del algoritmo siendo los coeficientes A, A', B, B', C y C' expresiones reales positivas que dependen linealmente de los tiempos.

Factores que dependen del tiempo de ejecución de un algoritmo:

1. Tamaño de los datos de entrada, simboliza la longitud n del vector.
2. El contenido de los datos de entrada deben oscilar entre los valores Tmin y Tmax.
3. El código generado por el compilador y el ordenador usado, afectan a los tiempos.

Cabe destacar que para analizar la eficiencia del algoritmo en el caso peor es fijar un tamaño del problema y desarrollar un análisis de la eficiencia en el caso

promedio. Para ello es necesario conocer el tiempo de ejecución y la frecuencia con que se presenta cada uno de ellos.

El análisis en el caso promedio se estudia mucho menos que el análisis en el caso peor, siempre realizaremos un estudio de la eficiencia del caso peor. La potencia del ordenador y eficiencia del compilador empleados en la ejecución del algoritmo. Recibe el nombre de criterio asintótico y se justifica de la siguiente manera:

- Analizar la eficiencia de los algoritmos de una manera independiente de las máquinas y lenguajes existentes. Cabe hacer hincapié en la diferencia de un programa y un algoritmo. Un programa es una implementación concreta y un algoritmo sujeta una serie de limitaciones tales como tamaño de la memoria, de los enteros que es posible representar, etc.

- Distintas implementaciones de un mismo algoritmo diferirán en sus tiempos de ejecución a lo sumo en una constante multiplicativa positiva, para tamaños del problema mayores. Si $t_1(n)$ y $t_2(n)$ son los tiempos de dos implementaciones, existe un real positivo c y un natural n_0 tales que para todo tamaño $n \geq n_0$ se cumple $t_1(n) \leq ct_2(n)$.

Para medir la eficiencia de un algoritmo es el tamaño n del problema. Es fácil determinar quién es dicha n en cada problema concreto. Cuando la entrada son vectores o ficheros, n sería de la mayoría de las veces la longitud de los mismos; si son matrices cuadradas, su dimensión; etc. en el caso de los algoritmos numéricos, es frecuente que n represente el valor de la entrada en lugar de su longitud.

El programador encontrará mejores algoritmos y no en la capacidad de cálculo de los ordenadores, analizará cómo depende del tamaño del problema el coste de los algoritmos, ignorando las constantes multiplicativas. Ciertas dependencias reciben nombres propios: lineal si es proporcional a n, cuadrática si es proporcional a n^2, etc. Estudiará el caso peor.

La memoria ocupada por un algoritmo crece lineal o cuadráticamente con el tamaño del problema. La eficiencia en tiempo y en espacio son objetivos contrapuestos. A veces se puede mejorar el tiempo de ejecución a costa de incrementar el espacio ocupado por el algoritmo o viceversa. El programador en función del algoritmo y recursos disponibles elegirá el más adecuado entre ambas magnitudes.

Órdenes de complejidad

Los órdenes son $O(\log n)$, $O(n)$, $O(n \log n)$, $O(n^2)$, $O(n^3)$ y $O(2^n)$. Los que se comportan de un modo más acorde con las expectativas de un usuario no informático son los de complejidad lineal y $O(n \log n)$: al duplicar el tamaño del problema se duplica aproximadamente el tiempo empleado, y al duplicar el tiempo disponible, el tamaño que es posible tratar también se duplica. El

algoritmo de O(log n) tiene un comportamiento bueno: doblar el tamaño del problema apenas afecta al tiempo de ejecución, mientras que doblar el tiempo disponible permite tratar problemas enormes en relación con el original.

Los órdenes $O(n^2)$ y $O(n^3)$ tienen un comportamiento claramente inferior al lineal. Dado un algoritmo del orden de $O(n^a)$, multiplicar el tiempo disponible es decir la velocidad del ordenador por un factor k multiplica el tamaño del problema que es posible tratar por un factor $\sqrt[a]{K}$.

Estos cálculos llevan a la conclusión de que no es posible tratar problemas demasiados grandes con algoritmos con esta tasa de crecimiento. Los algoritmos cuyas tasas de crecimiento están acotadas superiormente por n^a para cualquier a se dice que es de complejidad polinomial y los problemas que lo resuelven se llaman problemas tratables.

Utilizaremos algunas reglas para el cálculo de la eficiencia:

- Las instrucciones de entrada/salida o las expresiones aritméticas, siempre que no involucren variables estructuradas u operandos aritméticos cuyos tamaños dependen del tamaño n del problema se le asigna un coste en Θ(1). Donde Θ(f(n)) conjunto de las funciones.

- Para calcular el coste de una composición secuencial de instrucciones. Aplicamos la regla de la suma:

  $$\Theta(f(n)) + \Theta(g(n)) = \Theta(f(n) + g(n)) = \Theta(\max(f(n), g(n))).$$

  Si el coste de S_1 está en $\Theta(f_1(n))$ y el de S_2 está en $\Theta(f_2(n))$, entonces el coste de S_1; S_2 está en $\Theta(f_1(n)) + \Theta(f_2(n)) = \Theta(\max(f_1(n), f_2(n)))$.

- Para calcular el coste de una instrucción condicional

  Si B entonces S_1
  Si no S_2
  Fin Si

  Se tratarán las cotas superiores e inferiores. Si el coste de evaluar B está en $O(f_B(n))$, el de S_1 en $O(f_1(n))$ y el de S_2 en $O(f_2(n))$, entonces el coste de la instrucción condicional está en:

  $$\Theta(\max(f_B(n), f_1(n), f_2(n))).$$

  Siempre estamos suponiendo un análisis en el caso pero si no fuese así habría que estimar con qué frecuencia ejecuta el algoritmo S_1 y S_2 y realizar el correspondiente promedio f de f_1 y f_2 entonces el coste estaría en:

  $$\Theta(\max(f_B(n), f(n))).$$

  Calcular el coste de una instrucción iterativa:

Mientras B hacer S
Fin Mientras

Aplicamos la regla del producto que dice:

$$\Theta(f(n)) \cdot \Theta(g(n)) = \Theta(f(n) \cdot g(n))$$

Si el coste de evaluar B, más de ejecutar S, está en $O'(f_{B,S}(n))$ y el número de iteraciones es una función de n en $O'(f_{iter}(n))$, el coste total del bucle estará en $O'(f_{B,S}(n) \cdot f_{iter}(n))$. Si el coste de una iteración varía mucho de unas a otras entonces habría que realizar una suma desde 1 hasta $f_{iter}(n)$ de los costes individuales.

- Instrucciones más internas: es una instrucción condicional que solo tiene una asignación simple. Los costes de evaluar la expresión y de la asignación son ambos constantes $O'(\max(1, 1, 1) = O'(1)$. En definitiva el coste estará en $\Theta(1)$.

- Evaluación de la condición del bucle para más interno es decir la expresión $j \leq i + 1$, está también en $\Theta(1)$. La función que describe el número de iteraciones de este bucle es $f_{iter} = n - i$. Por lo tanto el coste del bucle interior se encuentra en $O'(n - i)$ y puede también estar en:

  $\Theta(n - i)$.

  Las instrucciones y la condición de control del bucle para más externo tienen todo un coste en $\Theta(1)$. Como coste de cada iteración del bucle más externo una expresión en $\Theta(n - i)$.

- Para calcular el coste total, hemos de sumar el coste de cada iteración del bucle externo, obteniendo la siguiente expresión:

  $$\sum_{i=1}^{n-1} n - i = \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i = (n-1)n - \frac{n(n-1)}{2}$$

  La expresión da un polinomio de grado n^2, es decir $\Theta(n^{2)}$.

Si hacemos uso del concepto de instrucción crítica, se denomina a la instrucción elemental que más veces se ejecuta dentro de un programa.

Obteniendo la expresión f(n) que calcula dicho número de veces se tiene directamente el orden de complejidad del algoritmo. Aquí aplicamos el criterio asintótico: para valores de n suficientemente grandes, el coste del resto del programa perderá importancia frente al coste de repetir f(n) veces la instrucción crítica por muy elemental que esta sea. El criterio asintótico es una herramienta muy útil para comparar en primera aproximación, la eficiencia de distintos algoritmos, pero tiene sus limitaciones:

- En la teoría algunos algoritmos esconden una constante multiplicativa muy grande que en la práctica son preferibles algoritmos teóricamente ineficientes. Como ejemplo a esta citación mencionaremos algoritmos para multiplicar matrices que empiezan a ser más rápidos que el

tradicional de complejidad del orden $\Theta(n^3)$, siendo n* n la dimensión de la matrices, tomando para n valores mayores que 500.

- Si un programa se va a utilizar muy poco puede dar como resultado un algoritmo más ineficiente, pero más rápido de desarrollar, que otro mejor pero más complicado. Cabe destacar que el coste de un programa incluye no sólo su explotación, sino también su desarrollo y mantenimiento posterior.

- La ganancia en el tiempo de ejecución se produce a costa de incrementar el espacio ocupado. Este coste puede ser inaceptable o puede implicar el uso de memoria externa, en cuyo caso los costes en tiempo habría que rehacerlos con otras constantes multiplicativas mucho más grandes.

El programador debe considerar los anteriores aspectos (tamaño de los problemas, frecuencia de uso del programa, recursos disponibles o deseables, coste de desarrollo, mantenimiento, etc.) y adoptar una solución que permita un compromiso entre todos ellos.

Análisis de la eficiencia de un programa recursivo

El análisis de la eficiencia de un programa recursivo es frecuente la aparición de funciones de coste también recursivas, es decir expresiones del tipo T(n) = E(n), donde en la expresión E, puede aparecer la propia función T. estas ecuaciones reciben el nombre de relaciones recurrentes. Resolver una recurrencia consiste en encontrar una expresión no recursiva para la función T(n).

La resolución de recurrencias no es tarea sencilla. Las ecuaciones e inecuaciones obtenidas en el análisis de un programa complejo pueden hacer su tratamiento matemático inmanejable.

Para reducir el problema mencionaremos dos tipos de reducción:

- Reducción del problema mediante sustracción

  Este tipo de recurrencia aparece al calcular el coste de programas recursivos en que el tamaño del problema decrece en una cantidad constante de una activación recursiva a la siguiente.

- Reducción del problema mediante la división

  Este tipo de recurrencias es típico de los programas recursivos que responden al esquema divide y vencerás. Una llamada al programa con un problema de tamaño n genera llamadas recursivas con subproblemas de tamaño n/b. Es decir cada subproblema tiene un tamaño que es un fracción del tamaño original. Resueltos los subproblemas, se combinan sus soluciones para producir la solución del problema original.

Diseño y verificación de programas recursivos

Diseño sistemático de programas y para su verificación formal, seguiremos restringiéndonos al caso de funciones recursivas lineales. Consta de los siguientes aspectos:

- Especificación formal del algoritmo: la especificación mediante predicados es aplicable indistintamente algoritmos recursivos e iterativos.

- Análisis por casos y escritura del programa: Se estudia cómo se puede descomponer recursivamente los datos del problema, de forma que se pueda calcular la solución pedida a partir de la solución al propio problema para datos más pequeños. Una solución eficiente se alcanza siempre escogiendo la descomposición recursiva que más drásticamente reduzca el tamaño del problema.

- Verificación formal de la corrección

$Q(\bar{x}) \Longrightarrow B_t(\bar{x}) \vee B_{nt}(\bar{x})$
1. $Q(\bar{x}) \wedge B_{nt}(\bar{x}) \Longrightarrow Q(s(\bar{x}))$
2. $Q(\bar{x}) \wedge B_t(\bar{x}) \Longrightarrow R(\bar{x}, triv(\bar{x}))$ (Base de la inducción)
3. $Q(\bar{x}) \wedge B_{nt}(\bar{x}) \wedge R(s(\bar{x}), \bar{y}') \Longrightarrow R(\bar{x}, c(\bar{y}', \bar{x}))$ Paso de inducción donde
4. $R(s(\bar{x}), \bar{y}')$ representa la hipótesis de inducción).
5. Encontrar t: $D_{T1} \rightarrow Z$ tal que $Q(\bar{x}) \Longrightarrow t(\bar{x}) \geq 0$ (definición de la estructura de pbf).
6. $Q(\bar{x}) \wedge B_{nt}(\bar{x}) \Longrightarrow t(s(\bar{x})) < t(\bar{x})$. (El tamaño de los subproblemas decrece estrictamente)

Tabla 171.Puntos a demostrar para verificar la corrección de f. Adaptado de [Ricardo Peña Mari]

Los puntos 1 y 2 son condiciones técnicas y expresan que f está bien definida. Los puntos 5 y 6 definen la estructura de pbf en D_f y demuestran que las llamadas recursivas se hacen sobre los datos $\{D_i\}_{i \in I}$ que forman una secuencia estrictamente decreciente. Los puntos 3 y 4 realizan la demostración de la corrección por inducción noetheriana sobre D_f.

El principio de inducción noetheriana se denomina también principio de inducción completa sobre preórdenes bien fundadas. Sea $(D, \preccurlyeq)$ un pbf y P(x) un predicado sobre los elementos x de D. Si es posible demostrar que todos los predecesores estrictos b de cualquier elemento a de D cumple el predicado P, también lo cumple el propio a, entonces todos los elementos lo cumplen: Formalmente.

$$(\forall a \in D.(\forall b \in D.b \prec a \rightarrow P(b)) \rightarrow P(a)) / (\forall a \in D.P(a))$$

Estudio de la eficiencia: Se reduce la magnitud que mide el tamaño del problema. El cual hemos explicado con detalle en las secciones anteriores.

4.3.7.11.2 Elementos de diseño

Los lenguajes de programación son mecanismos notacionales que se utilizan para instrumentar productos de la programación. Las características disponibles en el lenguaje de instrumentación ejercen una fuerte influencia sobre la estructura arquitectónica y los detalles algorítmicos del producto de la programación.

Los lenguajes de programación modernos proporcionan una variedad de características para apoyar el desarrollo y mantenimiento de los productos de programación. Estas características incluyen:

- Verificación robusta de tipos.
- Compilación por separado.
- Tipos de datos definidos por el usuario.
- Encapsulado de datos.
- Reglas de alcance flexibles genéricas.
- Manejo de excepciones definidas por el usuario.
- Mecanismo de concurrencia.

Estas características mencionadas anteriormente se analizarán mediante los siguientes lenguajes de programación como son:

- Pascal: Fue el primer lenguaje de programación moderno, sin embargo no soporta todas las características mencionadas anteriormente. Los objetivos principales de diseño para Pascal fueron que constituyeran una herramienta adecuada para la enseñanza de la programación estructurada y fuera fácil y eficientemente implantable en una diversidad de arquitecturas de máquina.

  Las características principales de pascal incluyen construcciones estructuradas para especificar la secuencia de ejecución de un programa, las facilidades para estructurar los datos y las reglas para verificar los tipos. Pascal tiene muchas deficiencias como lenguaje de programación para instrumentar productos de programación es decir no se proporcionan áreas de datos estáticas, no se permite dimensionar variables de arreglos, no se provee compilación por separado de módulos, no hay mecanismo para manejo de excepciones, las facilidades para el manejo de cadenas de caracteres y para entrada/salida son débiles.

- Ada: El lenguaje Ada se desarrolló bajo patrocinio del Departamento de Defensa de los Estados Unidos para apoyar el desarrollo de productos de la programación para sistemas de computación “empotrados”. Un sistema de computación empotrada es un componente de un sistema mayor y provee funciones de computación, comunicación y control para el sistema.

  Los sistemas empotrados son instrumentados utilizando microprocesadores y con frecuencia incorporan concurrencia y

procesamiento en tiempo real guiado por interrupciones. Ada no sólo incorpora características especiales para sistemas empotrados (manejo de interrupciones externas, acceso a detalles de representación de datos, bajo nivel de entrada/salida), sino que también proporciona características avanzadas para apoyar la programación en una amplia variedad de aplicaciones. Estas características son:

- Encapsulado de datos.
- Manejo de excepciones
- Mecanismo de concurrencia hacen de Ada un lenguaje de interés para una comunidad más amplia que el área de sistemas empotrados para la cual fue desarrollado.

Las diversas características de los lenguajes modernos pueden usarse para mejorar la calidad de la programación e incrementar la productividad de los programadores. Por ejemplo, la verificación robusta de datos mejora la calidad de un programa atrapando errores en la declaración y uso de entidades de datos a través de una combinación de revisiones al tiempo de compilación, tiempo de carga, tiempo de ejecución.

En Ada y otros lenguajes de programación modernos, los mecanismos tales como la conversión explícita de tipos, sobrecarga de subprogramas, sobrecarga de operadores, tipos de datos definidos por el usuario, abstracción de datos, rutinas genéricas y reglas de alcance flexibles, proveen la flexibilidad de programación requerida dentro del marco de trabajo de la verificación robusta de tipos, y así se proporciona tanto flexibilidad como seguridad.

Verificación de tipos de datos

Un tipo de dato especifica un conjunto de objetos de datos y un conjunto de operaciones permitidas sobre los objetos de ese tipo. Los objetos de tipo entero comprenden un rango de valores enteros dependientes de la instrumentación y un conjunto de operadores relacionales y aritméticos sobre literales y variables de tipo de entero.

El propósito de especificar tipos de datos es permitir la clasificación de los objetos de acuerdo a un uso deseado; permitir al traductor del lenguaje seleccionar las representaciones de memoria para los objetos de tipos diferentes y en el caso de lenguajes con verificación robusta de tipos, detectar y prevenir operaciones entre objetos de tipos distintos.

La verificación de tipos de datos se refiere a las restricciones y limitaciones impuestas a los modos en que los datos elementales pueden ser manejados por el programa. Distintos lenguajes imponen restricciones distintas.

Niveles de verificación de tipos de datos

- Nivel 0: Sin tipos

  Los lenguajes sin caracterización de tipos son construidos para áreas de aplicación específicas y normalmente tienen una utilidad limitada en otras aplicaciones.

- Nivel 1: Coerción automática de tipos

  El enfoque para la verificación de tipos es convertir automáticamente los operandos de tipos incompatibles, permitiendo así que la mayor parte de las operaciones ocurran entre operandos de diferentes tipos. La coerción automática de tipos proporciona máxima flexibilidad al programador y un máximo de sorpresas cuando se comete un error de programación. No hay seguridad en el manejo de los datos.

- Nivel 2: Modos Mixtos

  Fortran permite operaciones en modo mixto entre datos de tipo similares. La conversión la maneja automáticamente el traductor del lenguaje. La diferencia entre la coerción automática de tipos y la conversión en modo mixto es una cuestión de grado y no de clase. La mayor debilidad de la verificación de tipos en Fortran es la falta de verificación de las interfaces entre unidades del programa.

- Nivel 3: Verificación robusta de tipos

  La verificación robusta de tipos sólo permite operaciones entre objetos de tipos equivalentes. La verificación de tipos se logra utilizando revisiones al tiempo de compilación sobre los operadores y operandos y revisiones al tiempo de carga sobre las interfaces entre unidades de compilación.

  Las revisiones al tiempo de ejecución requieren verificar propiedades dinámicas tales como restricciones de subrango, uso apropiado de registros de variables y límites apropiados en subíndice de arreglos.

- Nivel 4: Verificación ligera de tipos

  La verificación robusta de tipos sólo permite operaciones entre objetos de datos de tipo equivalentes. Pascal ejemplifica la verificación ligera de tipos aunque la verificación es robusta en principio y exhibe escapatorias en la verificación de tipos en el tiempo de traducción, en el tiempo de carga y en tiempo de ejecución.

  La verificación de los tipos de los parámetros de funciones y procedimientos pasados como parámetros a otras funciones y procedimientos podría verificarse al tiempo de traducción, pero Pascal no requiere que se especifiquen esos parámetros.

Así no puede verificarse la corrección de los parámetros de procedimientos; no hay verificación en tiempo de carga de la concordancia en los tipos y número de parámetros para las rutinas compiladas independientemente. Al tiempo de ejecución, un registro variable puede adquirir dinámicamente diferentes variaciones en diferentes puntos dentro del programa.

Compilación por separado

La capacidad para desarrollar módulos y subsistemas, compilarlos y almacenarlos en biblioteca, y que el cargador automáticamente acceda a las unidades de la biblioteca y las ligue dentro del código objeto de un sistema, proporciona un poderoso mecanismo de abstracción.

Los módulos recompilados pueden utilizarse como componentes funcionales de un sistema, y la compilación por separado permite que diferentes programadores desarrollen o modifiquen simultáneamente diferentes módulos de un sistema. Ada soporta el concepto de compilación por separado.

Bajo la compilación por separado, la verificación de tipos a través de los límites de las unidades de compilación por separado debe ser tan segura como la verificación de tipos dentro de una unidad de compilación.

Definición de tipos por el usuario

Los lenguajes de programación tales como pascal y sus descendientes proporcionan una variedad de tipos de datos predefinidos, incluyendo enteros, punto flotante booleano, carácter, cadena de caracteres, apuntador, registros, arreglos y archivos. No todos los lenguajes proveen todos los tipos.

En lugar de eso, se puede predefinir unos cuantos tipos y proporcionar los mecanismos para que el usuario pueda utilizar otros tipos en términos de los tipos ya existentes. Hay las siguientes razones para proveer tipos de datos definidos por el usuario:

- Permitir la especificación de tipos que se necesitan con frecuencia en términos de tipos ya existentes.
- Permitir el mapeo de conceptos del dominio del problema al lenguaje de instrumentación.

Los tipos de datos definidos por el usuario pueden usarse para segmentar y modelar el dominio del problema. Esto puede mejorar la claridad y la seguridad de un programa de ordenador.

El mejoramiento en la claridad y seguridad facilita las tareas de depuración, prueba, documentación y modificación de un programa. El uso de los tipos de

datos definidos por el usuario es un fuerte indicador de la buena calidad de un programa.

Ada proporciona varios mecanismos que permiten al programador definir nuevos tipos de datos entre los cuales se encuentran:

Tipos de datos	Descripción
Subtipos	Un subtipo pone restricciones sobre el conjunto de posibles valores para un objeto de un tipo dado, pero no modifica el conjunto de operaciones permitidas. En Ada hay cuatro de clases de restricciones de subtipo: Intervalo: Hereda el conjunto de operaciones del tipo padre, pero el conjunto permitido de valores se restringe al subconjunto de los valores del padre Índice: Se usa para especificar los rangos de índices de arreglos. Precisión: Se utiliza para especificar la precisión de objetos de datos de punto fijo y punto flotante. Discriminación: Se utiliza para especificar los valores discriminantes permitidos para objetos de registros Se requieren verificaciones al tiempo de ejecución para determinar que los objetos satisfacen sus restricciones de subtipo, y la violación de una restricción hace que surja la condición de excepción.
Derivados	Se utiliza para declarar nuevos tipos para objetos distintos en el dominio del problema que tiene representaciones similares en el dominio de la solución.
Enumerables	Se define listando o enumerando el conjunto de valores para objetos de este tipo. Los valores enumerados se denotan por identificadores y literales de carácter. Ellos tienen la relación de orden especificadas en la enumeración; el número de posición del primer valor listado es cero, y cada valor sucesivo tiene un número de posición mayor en uno que el de su predecesor. Los literales enumerados pueden asignarse a variables del tipo correspondiente, el traductor del lenguaje mapea los literales enumerables sobre números de posición de 0 a (N-1), donde N es el número de literales en la definición del tipo. La entrada y salida de literales enumerados se permite en Ada y no en Pascal.
Registros	Es un objeto de datos compuesto. Los componentes de un registro reciben nombre y pueden ser de tipos distintos. Los registros pueden así adaptarse a las necesidades de una aplicación particular. Un conjunto de objetos de registro puede ser el componente de un archivo o un arreglo, de modo que el conjunto de todos los registros de personal puede ser referido y manipularse como una sola entidad. En la mayor parte de los lenguajes de programación modernos, los registros pueden adaptarse aún más para el uso de discriminantes y variables. Los discriminantes se pueden usar para parametrizar un tipo de registro.
	Los apuntadores son objetos que permiten el acceso a otros objetos. Los objetos designados por apuntadores se ubican dinámicamente durante la ejecución de un programa. Cuando se crea un objeto dinámico, el valor del apuntador asociado

Apuntadores	designa o apunta al objeto recién creado. En algunos lenguajes de programación un solo apuntador puede usarse para acceder a los objetos de diferentes tipos. Para permitir la verificación robusta de tipos de objetos dinámicos, la mayor parte de los lenguajes de programación modernos utilizan la técnica de Pascal de asociar tipos de apuntadores únicos con tipos de objetos distintos. Las estructuras dinámicas de datos definidas por el usuario de cualquier topología deseada pueden especificarse asociando tipo apuntadores con tipos registros e incrustando apuntadores dentro de los registros para permitir el acceso a otros registros. La posibilidad de especificar una topología arbitraria y de ubicar dinámicamente a los objetos hace útiles a estas estructuras cuando se necesita una gráfica dirigida de relaciones entre datos elementales, o cuando el número de elementos en una estructura de datos varía dinámicamente o cuando se requiere un acceso rápido a objetos de datos particulares. Las listas ligadas y los árboles binarios son ejemplos comunes de estructuras de datos dinámicos.
Abstracción de datos	El concepto de abstracción de datos incorpora: Encapsulado de datos: Difiere de los tipos de datos abstractos en que el encapsulado proporciona solamente un ejemplar de una entidad. Tipos de datos abstractos: Es una plantilla a partir de la cual se pueden crear múltiples ejemplares, en el mismo sentido en que el tipo de puntos flotante es una plantilla para la creación de múltiples objetos de punto flotante. Le permite al programador declarar plantillas para la parte visible y los detalles de representación y manipulación de los tipos de datos abstractos. La abstracción de datos proporciona un poderoso mecanismo para escribir programas bien estructurados y fáciles de modificar. Los detalles internos de la representación y manipulación de datos puede modificarse a voluntad, suponiendo que las interfaces de los procedimientos de manipulación permanezcan iguales; los otros componentes del programa no serán afectados por la modificación, excepto tal vez por las modificaciones en las características del funcionamiento y los límites de capacidad.
Facilidades genéricas	Permite la parametrización textual de una unidad de programa (en Ada, los procedimientos, funciones, paquetes y tareas son unidades de programas). La facilidad genérica es una poderosa herramienta de propósito general que puede usarse en muchos contextos distintos. Las cláusulas genéricas son una extensión natural de la parametrización de subprogramas. En general, una facilidad genérica, tal como las que proporciona Ada, permite la factorización de propiedades comunes de las unidades de programa. Esto da como resultado una sola copia de la unidad de programa, lo que a su vez promueve la modularidad y puede dar como resultado una mayor eficiencia del traductor, mientras que se apoya la verificación robusta de tipos.

Tabla 172.Tipos de datos

4.3.7.11.3 Procesamiento de atributo

Una descripción de las reglas utilizadas por la entidad para cumplir su función. El atributo de proceso describe el algoritmo utilizado por la entidad para realizar una tarea específica y sus contingencias.

La técnica de la codificación estructurada es linealizar el flujo de control a través de un ordenador, de modo que la secuencia de ejecución siga a la secuencia en que está escrito el código. La estructura dinámica de un programa a medida que se ejecuta se parece entonces a la estructura estática del texto escrito.

Esto mejora la legibilidad del código, lo cual facilita la comprensión, depuración, prueba, documentación y modificación de programas. También facilita la verificación formal de programas. El flujo de control lineal se puede lograr restringiendo el conjunto de construcciones de programas permitidas a formatos de entrada, salida. A continuación se analizan los siguientes aspectos:

- Una entrada, una salida

  La secuenciación, la selección entre acciones alternativas, y la iteración forman un conjunto suficiente de construcciones para describir el flujo de control del algoritmo. Cualquier segmento de programa que contenga una entrada y una salida en las preposiciones en algún camino de la entrada a la salida puede especificarse utilizando secuenciación, selección e iteración.

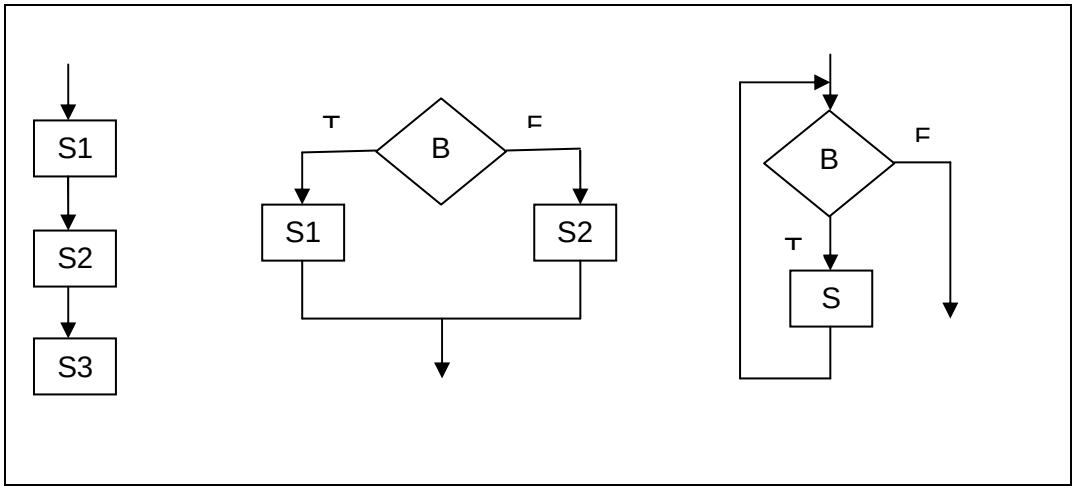


Figura 108.Diagrama de flujo para codificación estructurada

Las construcciones con una entrada y una salida para especificar el flujo del control en algoritmos son:

Secuenciación: S1, S2, S3.
Selección: Si B entonces S1 sino S2.

La propiedad de una entrada y una salida permite el anidamiento de construcciones en cualquier manera que se desee. Cada proposición S1 debe ser una proposición de asignación, una llamada a una función, un si-entonces-si no, o un while-hacer.

El aspecto más importante de la propiedad de una entrada y una salida es que se conserva la linealidad del flujo de control, aún con una profundidad arbitraria de anidamientos de construcciones.

- Consideraciones de eficiencia

  Las construcciones de una entrada, una salida, una crítica dan como resultado una utilización ineficiente del espacio de memoria y del tiempo de ejecución. Con frecuencia se cita la necesidad de variables auxiliares, segmentos de código repetido y una excesiva llamada a subprogramas.

- Incumplimiento de una entrada, una salida

  El objetivo de la codificación estructurada es mejorar la claridad y legibilidad de los programas fuentes. La claridad se mejora mediante la utilización de construcciones de una entrada, una salida. Hay situaciones que ocurren comúnmente que garantizan el incumplimiento de una entrada, salida. Tres situaciones que suceden generalmente son: salidas múltiples de ciclos, manejo de errores, encapsulado de datos. El mejor procedimiento para tratar con salidas múltiples de ciclos es rediseñar el algoritmo de modo que no se requieran las salidas múltiples. Una técnica para lograr las salidas múltiples de los ciclos es introducir una variable booleana para cada condición y probarla en cada iteración del ciclo. La técnica mencionada anteriormente presenta los siguientes problemas:

  - Necesidad de variables adicionales.
  - Las pruebas adicionales que se requieren a la salida del ciclo para determinar cuál condición causó la salida.
  - Requisito, en muchos casos el cuerpo del ciclo se termine inmediatamente después de la ocurrencia de una condición dada y no al inicio de la siguiente iteración como ocurrirá al probar una condición en el ciclo.

  Una solución a este problema es utilizar goto, el cual el lenguaje de programación FORTRAN lo soporta. Goto se utiliza para lograr las salidas múltiples del ciclo y para aislar las acciones alternativas en la salida, se mantiene la claridad y localidad. Una proposición de salida en el lenguaje de programación Ada transfiere control a la proposición que sigue inmediatamente al final del ciclo que la contiene cuando la condición asociada se vuelve verdadera.

- Encapsulado de datos

  El encapsulado de datos implica el empaquetamiento de una estructura de datos y sus rutinas de acceso, y otras rutinas utilizan la estructura de datos llamando a las rutinas de acceso apropiadas. La estructura de datos se define por las operaciones que se pueden realizar sobre ella. Las rutinas que utiliza la estructura de datos no necesitan conocer los detalles de la representación o manipulación de datos.

  El encapsulado de datos no debe confundirse con los tipos de datos abstractos ni con la abstracción de datos. Un tipo de datos abstractos es una plantilla definida por el usuario que puede utilizarse para crear objetos de datos encapsulados. El encapsulado de datos se refiere a un solo ejemplar de un objeto de datos abstractos. La abstracción de datos se utiliza para describir un principio de diseño que incorpora el encapsulamiento de datos y a los tipos de datos abstractos.

  En los lenguajes de programación primitivos, el encapsulado de datos se puede instrumentar en una forma que incumple la regla de una entrada, una salida y aún mantiene la linealidad y localidad del flujo de control.

- Preposición Goto

  Proporciona la transferencia de control incondicional y de esta manera permite el incumplimiento de la condición de una entrada, una salida. La preposición goto se utiliza también para la construcción de una entrada, una salida en lenguajes de programación primitivos.

  Una preposición goto debe reservarse para situaciones raras o poco comunes donde tiene que romperse la estructura natural de un algoritmo. Una buena regla es evitar el uso de saltos para expresar la iteración regular o la ejecución condicional de preposiciones, pues tales saltos destruyen el reflejo de la estructura de la computación en la estructura textual (estática) del programa.

  La falta de correspondencia entre la estructura textual y la computacional (estática y dinámica) es extremadamente perjudicial a la claridad del programa y hace mucho más difícil la tarea de verificación. En los lenguajes de programación primitivos, la versión goto puede ser la mejor estructura que se puede lograr.

  Una estructura preferible es la forma más compacta:

```
Si (condición 1) entonces
        Código para la condición 1
    Else si(condición 2) entonces
        Código para la condición 2
    Else
        Código para la condición 3
Fin Si
```

La preposición goto puede ser un mecanismo valioso cuando se utiliza de manera disciplinada y con estilo. Las proposiciones goto que transfieren el control a regiones remotas del código, o que saltan adentro y afuera de segmentos de código en formas complicadas, destruyen la linealidad y localidad de flujo de control. Deben evitarse.

- Recursividad

  Un subprograma recursivo es uno que se llama a sí mismo, ya sea directamente o indirectamente; la técnica de recursividad es muy eficaz y a la vez elegante cuando se utiliza adecuadamente, los subprogramas recursivos son fáciles de entender y eficientes. Mediante la utilización adecuada de la recursividad se cumplen los objetivos de claridad y eficacia. La recursividad es una técnica de especificación y en circunstancias apropiadas una técnica de instrumentación.

  Una especificación recursiva no necesariamente implica una instrumentación recursiva. Las especificaciones funcionales establecen qué se requiere, no cómo alcanzarlo. La forma algorítmica de una especificación recursiva a menudo se expresa mejor en una manera iterativa.

  Una instrumentación iterativa de un algoritmo recursivo da como resultado una pila manejada por el usuario para simular la recursividad o un manejo complicado de apuntadores en la estructura de datos para establecer las ligas de encadenamiento hacia atrás. Esto a veces es menos eficiente y siempre es menos compresible que una instrumentación recursiva.

4.3.7.11.4 Ejemplos

Seudocódigo

El seudocódigo puede reemplazar los diagramas de flujo y reducir una buena cantidad de documentación externa requerida para describir al sistema

```
INICIAR tablas y contadores; ABRIR ficheros
LEER el primer registro del texto
   MIENTRAS se encuentren registros de textos REPETIR
    MIENTRAS haya más palabras en el registro de texto REPETIR
      EXTRAER  la siguiente palabra
      BUSCAR en la tabla de palabras la palabra extraída
      SI la palabra extraída se encontró ENTONCES
        INCREMENTAR el contador de ocurrencias de la palabra extraída
      SI NO
        INSERTAR la palabra extraída en la tabla de palabras
      FIN SI
```

```
    INCREMENTAR el contador de palabras procesadas
  FIN MIENTRAS al final del registro del texto
FIN MIENTRAS cuando todos los registros han sido procesados
IMPRIMIR la tabla de palabras y el contador de palabras procesadas
CERRAR fichero
Fin del programa
```

Tabla 173: Diseño en seudocódigo

4.3.7.12 Conclusiones

Destacaremos los siguientes puntos:

- El diseño de datos es la primera de las actividades de diseño y por consiguiente tiene una gran importancia, ya que la estructura de datos repercute en la estructura del programa y por supuesto en la complejidad procedimental y por ende en la calidad del producto final.

- Se ha analizado una amplia variedad de anotaciones que pueden ser utilizadas para apoyar y documentar el desarrollo de un modelo de diseño. Los ejemplos seleccionados ilustran los formularios utilizados por la caja negra y la caja blanca y el modelo en el conjunto de puntos de vista (funcional, los datos de comportamiento, modelado y construcción). La elección de una notación en particular, por supuesto, dependerá de muchos factores, incluyendo el dominio del problema, las ideas sobre la forma arquitectónica del sistema final, y también las prácticas de diseño especial que se utilizan.

- A pesar de la relativa simplicidad y elegancia de las ideas básicas que conforman el modelo orientado a objetos, y su dominio en términos de mecanismos de aplicación, sigue proporcionando un importante desafío para el diseñador y para la transferencia de conocimientos de diseño entre los diseñadores. Los patrones de diseño, marcos y todos los métodos tienen algo que aportar y, de hecho, el posible diseñador de sistemas orientados a objetos realmente requiere un cierto grado de familiaridad.

- Hemos descrito un conjunto de métricas para diseños orientados a objetos en el cual se han proporcionado guías como medio para producir sistemas de calidad y problemas al analizar las métricas como son: definiciones ambiguas que dan lugar a distintas interpretaciones, falta de concordancia para el propósito en el cual fueron concebidas, falta de validez teórica.

- La importancia que tiene disponer de métricas para evaluar la calidad de la base de datos, bases que se han convertido en los últimos años en el componente central de los sistemas de información.

4.3.8 Conclusiones generales

- Hemos dedicado en este capítulo requisitos, análisis y diseño sobre los cuales UML hace mayor hincapié. UML es un lenguaje que desarrolla dibujos en sus objetivos que son comparables a los esquemas que se utilizan hace tiempo en otras disciplinas técnicas. Estas áreas de mayor énfasis en las que el proceso desarrolla sistema de software completo de acuerdo a la arquitectura.

- Se han analizado los conceptos fundamentales de diseño incluyendo abstracción, cobertura de información, estructura, modularidad, concurrencia.

- La orientación de procesos en el desarrollo del software y por ende la necesidad de mejora continua de las actividades y tareas que se realizan durante estos procesos han llevado a realizar investigaciones en este campo y la aparición de algunos modelos de evaluación y mejora de los procesos del ciclo de vida del software.

- La consistencia de un modelo de procesos en una empresa puede suponer una mejora interna o un reconocimiento que ofrezca una ventaja competitiva en la adjudicación de nuevos proyectos. Estos dos factores son cada vez tenidos en cuenta en el sector del desarrollo del software y hoy en día se está produciendo un incremento notable en el interés por estas buenas prácticas.

- Durante el transcurso de este capítulo hemos hecho referencia al proceso unificado, este proceso de desarrollo es de gran importancia porque reúne un conjunto de actividades que son necesarias para transformar los requisitos de un usuario en un sistema software, además de ser un proceso abarca aspectos tales como: marco de trabajo genérico en una gran variedad de sistemas software, en distintas áreas de aplicación, distintos tipos de organizaciones, distintos niveles de aptitud, distintos tamaños del proyecto.

- Hoy por hoy la tendencia actual del software lleva a la construcción de sistemas grandes y además complejos. Esto se debe en parte a que los ordenadores cada vez son más potentes y por ende el usuario espera más de ellos. Esta tendencia se debe también al uso creciente de Internet que conlleva el intercambio de todo tipo de información. La mejora continuada de una versión con otra crece a medida que vemos como pueden los productos software ser mejorados. Nuestro empeño de tener un software que esté adaptado a nuestras necesidades esto con lleva a tener un software más complejo.

- La presencia de un proceso bien definido y además correctamente gestionado es una distinción esencial entre proyectos hiperproductivos y otros que fracasan.

- Los conceptos de calidad del software se ocupan de evaluar tanto la estructura estática como el comportamiento dinámico del sistema final.

- Se ha de profundizar en investigación sobre la calidad del proceso, tanto del proceso de modelado como del proceso de obtención y carga de los datos, ya que la mayor parte de los estudios analizados se centran en la calidad del producto.

- Las descripciones formales pueden proporcionar una ayuda muy importante para el desarrollo de un diseño. Las técnicas de diseño necesarias para la derivación de una especificación formal a diferencia de las técnicas matemáticas están mucho menos desarrolladas. Esto deja abierta la pregunta de cuándo hacer uso de estas técnicas y en qué escala. Hay relativamente pocos documentos acerca del uso de métodos formales para el desarrollo de sistemas muy grandes y, de hecho, esta puede no ser la mejor manera de hacer uso de sus puntos fuertes. Hay evidencia de uso cada vez mayor para el desarrollo de sistemas de alta integridad o por lo menos, de aquellas partes de un gran sistema que pueden requerir alta integridad.

- Los conocimientos de diseño del software siguen siendo difíciles de alcanzar, en términos de conocimiento acerca de los productos, y también sobre los procesos utilizados para desarrollarlos.

4.4 Fundamentos de las pruebas del software. Adaptado del estándar IEEE 729-2008

4.4.1 Introducción

He tomado de referencia el estándar IEEE 729-2008, el propósito de esta norma está basado en pruebas del software, ayuda a la organización de desarrollo de la calidad a la construcción del software y al sistema durante los procesos del ciclo de vida y a comprobar que la calidad se ha logrado. La prueba del software determina si los productos de una actividad del ciclo de vida cumplen con las expectativas de los requisitos y además el producto cumple con el uso previsto y las necesidades del usuario.

Para ello he suscitado algunos aspectos de mayor importancia con respecto a esta norma adecuándolos al estudio del siguiente esquema:

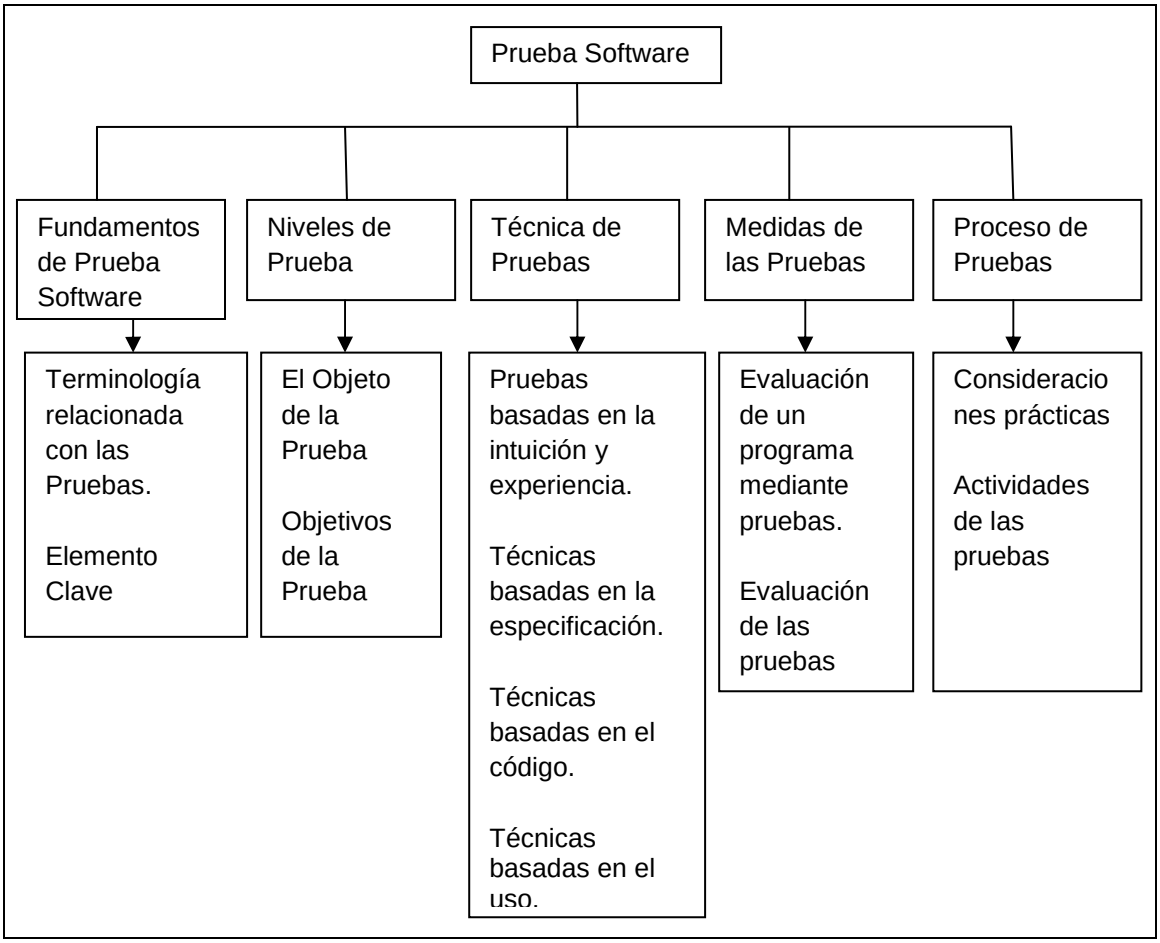


Figura 109.Este esquema se denomina: división de los temas para el KA de las Pruebas del Software. Documentación generada por Domigo Gaitero

He decidido optar por este esquema para su estudio porque me aporta mayor claridad, está orientado a la confiabilidad del software y por ende las pruebas del software es uno de los procesos fundamentales dentro del control de la calidad de software.

Los productos software están creciendo vertiginosamente en muchos aspectos de la vida cotidiana. Este crecimiento de complejidad conlleva la posibilidad de incrementar fallos y errores durante su uso en el cual pueden acarrear una serie de consecuencias en términos económicos, de tiempo o incluso de vidas humanas. Por ello es importante incluir actividades de aseguramiento de calidad durante todo el proceso de desarrollo y mantenimiento.

La complejidad del software causa que los procesos de prueba jueguen un papel primordial para aumentar la calidad del producto final, a lo largo de este capítulo se describen aspectos que están relacionados con la prueba del software.

Se detallan: terminología relacionada, cuestiones clave, relaciones de las pruebas con otras actividades, niveles de prueba, objeto de la prueba, objetivos de las pruebas, técnicas de pruebas basadas en: intuición y experiencia; en especificación; en código; uso; naturaleza de la aplicación, medidas de las

pruebas, evaluación de las pruebas realizadas, procesos de las pruebas, actividades de prueba y cómo gestionar los procesos de prueba a lo largo del proyecto. La prueba completa o exhaustiva es imposible para ello es fundamental seleccionar casos de prueba usando técnicas y herramientas.

La fase de las pruebas del software es un elemento crítico para la garantía de la calidad del software y representa revisiones de las especificaciones, del diseño y de la codificación. La importancia de las pruebas del software supone un 40% del coste total de desarrollo.

Cabe recalcar que en cada fase del ciclo de vida de desarrollo del software se realiza un conjunto de pruebas que permiten comprobar que el software de desarrollo satisface las especificaciones de cada fase.

4.4.2 Terminología relacionada con las pruebas

4.4.2.1 Definiciones de prueba y terminología

- Pruebas de aceptación

  Implican la planeación y ejecución de pruebas funcionales, de desempeño y de tensión para demostrar que el sistema implantado satisface sus requisitos. A menudo, se hacen correr en el programa dos conjuntos de prueba de aceptación:

  - Aquellas pruebas desarrolladas por el grupo de control de calidad.
  - Y las que desarrolla el cliente

  Además de las pruebas funcionales y de desempeño, las pruebas de tensión se realizan con el fin de establecer las limitaciones del sistema. Ejemplo, un compilador puede probarse para determinar el efecto del sobre flujo en la tabla de símbolos o un sistema de tiempo real se puede probar para conocer el efecto de muchas interrupciones de alta prioridad.

  Las pruebas de aceptación incorporan casos de prueba desarrollados durante las pruebas de unidad y de aceptación. Se añaden casos de prueba adicionales para lograr el nivel deseado de las pruebas funcionales, de desempeño y de tensión del sistema completo.

  Entre las herramientas de especial importancia durante las pruebas de aceptación se hallan un analizador de cobertura de la prueba y otro de tiempos y un verificador de los estándares de codificación. Un analizador de cobertura de la prueba registra las rutas de control seguidas por cada caso de prueba. El registro acumulado sirve para fijar la extensión de la cobertura de la prueba obtenida durante la prueba de aceptación.

Sin esta herramienta, sería imposible establecer la extensión lograda de la cobertura de prueba. En los grandes sistemas, el analizador de la cobertura puede registrar sólo las rutinas llamadas y no las proposiciones individuales ejecutadas. Un analizador de tiempos informa el tiempo empleado en varias regiones del código fuente bajo diferentes casos de prueba.

- Tolerante a fallos

  Pertenecientes a un sistema o componente que sea capaz de continuar la operación normal a pesar de la presencia de fallos.

- Modo de fallo

  Manifestación física o funcional de un fracaso. Por ejemplo, un sistema en el modo de fallo puede ser caracterizado por un funcionamiento lento, los resultados incorrectos, o la terminación completa de la ejecución.

- Tasa de fallos

  Relación entre el número de fallos de una categoría dada a una unidad de medida, por ejemplo, fallos por unidad de tiempo, fallos por número de transacciones, fallos por número de ejecuciones por equipo.

- Error

  Diferencia entre un cálculo, valor observado o medido o condición. Un paso incorrecto, proceso, o de definición de datos. Un resultado incorrecto. Acción humana que produce un resultado incorrecto.

- Error del modelo

  Evaluación de software, un modelo utilizado para estimar o predecir el número de fallos restantes, el tiempo de prueba necesario, y las mismas características de un sistema.

- Pruebas

  El proceso de funcionamiento de un sistema o componente bajo ciertas condiciones, observar o registrar los resultados, y hacer una evaluación de algún aspecto del sistema o componente. El proceso de análisis de un elemento de software para detectar las diferencias entre las condiciones existentes y necesarias y para evaluar las características de los elementos de software.

Anteriormente hemos mencionado algunas definiciones acerca de las pruebas del software; durante el transcurso de este tema se plasman más adelante con detalle conceptos de las pruebas del software.

4.4.2.2 Errores Vs fallos

Se define formalmente un error como la discrepancia entre un valor o condición calculado, observado o medido y el valor o condición específica o teóricamente correcta.

Se define formalmente un fallo: que comete el desarrollador.

Se define un defecto como la desviación de estándares, y finalmente, fallo es la consecuencia de un defecto. Por lo tanto estos dos términos tienden a la confusión y se utiliza uno de ellos de forma genérica.

Un error y un defecto software, y como consecuencia un fallo existen cuando el software no hace lo que el usuario espera que haga, es decir, aquellos que se han acordado previamente en la especificación de requisitos. En una gran parte de los casos, esto se produce por un error de comunicación con el usuario durante la fase de análisis de requisitos o por un error de codificación

4.4.3 Cuestiones clave

4.4.3.1 Criterios para dar por finalizadas las pruebas

Partiendo de la base, estadísticamente no hay ningún software entregado y dado por finalizado que esté libre de errores, sí es necesario que exista un alto porcentaje de confianza en el sistema desarrollado. Estadísticamente se acepta un 95% como porcentaje válido.

En cualquier caso, un buen diseño de caso de prueba ayuda a elevar el grado de confianza escogiendo aquellos casos de prueba que tengan más probabilidades de tener errores y cubran el espectro de pruebas de la forma más amplia posible.

4.4.3.2 Efectividad de las pruebas/Objetivos para las pruebas

En un conjunto de casos de prueba siempre pensamos qué tan efectivos son, pero tenemos que aclarar lo efectivo. La elección de un método se utiliza para generar casos de prueba, y luego ejecutar los casos de prueba. La conformidad se puede medir, por lo que se puede utilizar como base para el cumplimiento contractual.

Se requiere que los evaluadores elijan los métodos adecuados. Podemos obtener otra mejora incremental mediante la elaboración de métodos híbridos es una técnica estructural de rendimiento. El concepto de rutas de ejecución del programa, proporciona una buena formulación de la eficacia de la prueba.

Cuando una determinada ruta se recorre más de una vez, se podría cuestionar la redundancia.

La mejor interpretación de efectividad de un conjunto de casos de prueba es para detectar un fallo presente en un programa. Esto es problemático por dos razones:

- En primer lugar, se da por supuesto que sabemos todos los defectos en un programa. Pero afirmar este supuesto es un error dado que no sabemos todos los defectos en un programa, nunca podríamos saber si los casos de prueba a partir de un determinado método es el más propicio.

- La segunda razón es más teórica: Demostrar que un programa es libre de errores, esto es casi imposible. Lo mejor que podemos hacer es trabajar tipos de errores. Podemos optar por los métodos de prueba funcionales y estructurales que pueden revelar errores de ese tipo.

  Si unimos esto con el conocimiento de las formas más probables de los fallos nos encontramos con un enfoque pragmático de la eficacia de las pruebas. Se trata de mejorar si hacemos un seguimiento de las clases y frecuencias de los errores en el software que desarrollamos.

4.4.3.3 Realizar pruebas para la identificación de defectos

Una de las consideraciones más importantes en la aplicación del sistema de prueba es determinar quién debe hacerlo. Para responder a esto de una manera negativa, los programadores no deben realizar una prueba del sistema, y todas las fases de prueba, en que la organización responsable del desarrollo de los programas definitivamente no debe realizarse.

- El primer punto se deriva del hecho de que una persona que realiza una prueba del sistema debe ser capaz de pensar como un usuario final, lo que implica un conocimiento profundo de las actitudes del usuario final y de cómo el programa se utilizará.

  Obviamente, si es posible, un candidato a prueba es uno o más usuarios finales. Sin embargo, debido a que el usuario final típico no tiene la capacidad ni la experiencia para realizar muchas de las categorías de las pruebas descritas anteriormente, un equipo de sistema de prueba ideal podría estar compuesto por unos pocos expertos profesionales del sistema de prueba (las personas que pasan su vida en la realización de pruebas del sistema).

- El segundo punto se deriva del hecho de que una prueba del sistema es un "todo vale, no todo vale" de actividad. Una vez más, la organización de desarrollo tiene vínculos psicológicos con el programa que van en contra de este tipo de actividad.

Además, la mayoría de las organizaciones de desarrollo son las más interesadas en que la prueba del sistema proceda de la mejor manera posible y en la fecha prevista, y no están realmente motivados para demostrar que el programa no cumple con sus objetivos. Por lo menos, la prueba del sistema debe ser realizada por un grupo independiente de personas, vínculos con la organización de desarrollo o subcontratar las pruebas a una empresa independiente.

4.4.3.4 El problema del oráculo

Genera predicciones de resultados esperados de pruebas. Los oráculos pueden ser versiones previas del programa o sistemas de prototipos. Las pruebas back-to-back implican ejecutar el oráculo y el programa a probar en paralelo. Las diferencias entre sus salidas son resaltadas.

Un prototipo es una versión inicial de un sistema software que se usa para demostrar conceptos tales como: opciones de diseño a probar, informes de problemas y solventar las posibles soluciones. Un prototipo se utiliza de distintas maneras:

- En el proceso de requisitos, un prototipo fomenta la obtención y validación de los requisitos del sistema.
- En el proceso de diseño se puede utilizar el prototipo para analizar soluciones y fomentar el diseño de las interfaces de usuario.
- En el proceso de pruebas se puede utilizar el prototipo para ejecutar pruebas back-to-back con el sistema que se entregará al cliente.

Un problema importante en las pruebas del sistema es la validación de las pruebas, se comprueba si los resultados de una prueba son los esperados.

Cuando un prototipo del sistema está disponible, se reduce el esfuerzo realizado en la comprobación de los resultados ejecutando prueba vuelta a vuelta (back-to-back). La construcción de prototipos aumenta los costes en las etapas iniciales del software pero disminuye los costes posteriores en el proceso de desarrollo.

El principal discernimiento de esto es evitar rehacer el trabajo durante el desarrollo debido a que los clientes solicitan menos cambios en el sistema. *Gordon y Bieman [Gordon y Bieman,1995] observaron que el rendimiento general del sistema algunas veces se degrada si se reutiliza código ineficiente proveniente del prototipo.*

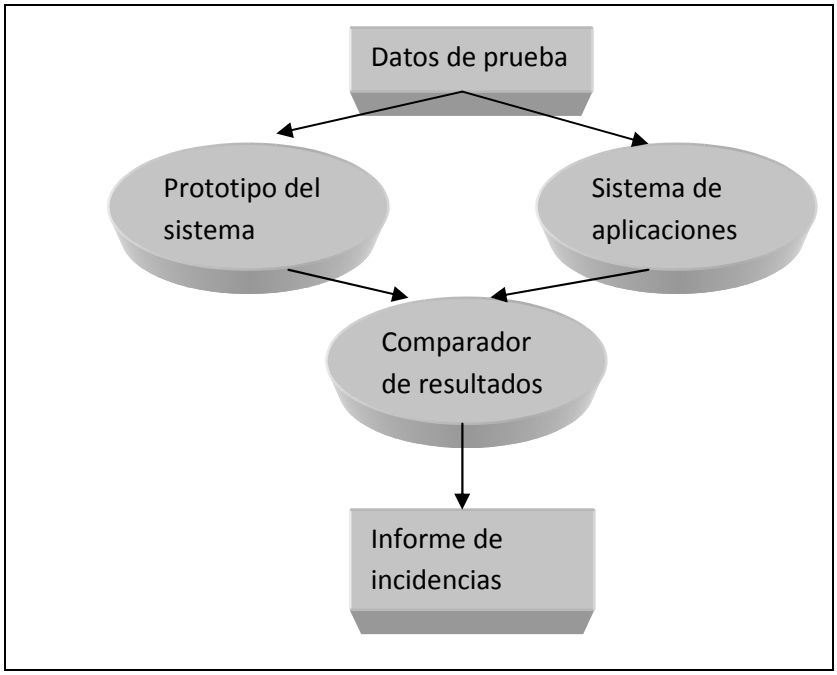


Figura 110.Pruebas back- to-back. Adaptado de [Ian sommerville]

4.4.3.5 Limitaciones teóricas y prácticas de las pruebas

Existen muchas aplicaciones en las cuales el dominio de entrada es relativamente limitado, es decir, el número de parámetros de entrada es pequeño y los valores que cada uno de los parámetros puede tomar están claramente acotados. Cuando ciertos números son pequeños, es posible determinar cada permutación de entrada y probar de manera exhaustiva el dominio de entrada. Conforme crece el número de valores de entrada y el número de valores discretos para cada elemento de datos. La prueba exhaustiva se convierte en impracticable e imposible.

Como práctica de las pruebas podemos mencionar la prueba de arreglo ortogonal, puede aplicarse a problemas en que el dominio de entrada es relativamente pequeño pero demasiado grande para alojar la prueba exhaustiva. El método de prueba de arreglo ortogonal es muy útil para encontrar los fallos de región de una categoría de error asociada con lógica defectuosa dentro de un componente de software.

Phadke valora el resultado de las pruebas utilizando las pruebas de arreglo ortogonal de la siguiente forma:

- Detectar y aislar todos los fallos de modo individual: Un fallo de modo individual es un problema congruente con cualquier nivel de cualquier parámetro individual.

- Detectar todos los fallos de modo doble: si existe un problema consistente cuando ocurre en un conjunto de niveles especificados de dos parámetros, se llama fallo de modo doble. Un fallo de modo doble es

indicio de incompatibilidad pareada o de interacción dañinas entre dos parámetros de prueba.

- Fallo multimodo: Solo puede garantizar la detección de fallos de modo individual y doble. Muchos fallos multimodo también son detectables por estas pruebas.

Podemos mencionar dos limitaciones en las herramientas de cobertura y herramientas de captura y reproducción que se describen con más detalle en el apartado 4.4.12.2.3 Desarrollo del entorno de pruebas.

- Herramienta de cobertura

  Una limitación importante de este tipo de herramienta es el tamaño, en cuanto al número de líneas de código, que son capaces de manejar. Debido a la instrumentación añadida, el tiempo dedicado a la ejecución de las pruebas puede ser excesivo en proyectos de tamaño medio y grande.

- Herramientas de captura y reproducción

  Su coste de implementación puede ser alto ya que requiere una versión muy alta en infraestructura que las soporte, tales como educación y entrenamiento en su uso, y en ciertos casos basadas principalmente en formularios.

4.4.3.6 Posibilidad de hacer pruebas

Una estrategia sistemática para probar el software, incluso la mejor estrategia fracasará si no se aborda una serie de aspectos decisivos:

Nombre	Descripción
Especifican los requisitos del producto en forma cuantificable mucho antes de comenzar con las pruebas	El objetivo de una prueba es encontrar errores, una estrategia de prueba valora otras características de calidad como: la portabilidad, el mantenimiento y facilidad de uso.
Establecen de manera explícita los objetivos de las pruebas	Los objetivos específicos de las pruebas deben anunciarse en términos medibles, es decir, la efectividad de las pruebas, cobertura, costos por descubrir y corregir etc.
Entienden a los usuarios del software y desarrollan un perfil para cada categoría de usuario	Los casos de uso describen el escenario de interacción para cada clase de usuario y reducen el esfuerzo de la prueba global en la utilización del producto
Construcción de software robusto	Diseñado para probarse a sí mismo. El software debe poder diagnosticar ciertas clases de errores.
Utilización de revisiones efectivas como filtro previo a las pruebas	Revisiones técnicas pueden ser tan eficaces como probar para descubrir

	errores. Por esta razón las revisiones disminuyen la cantidad del esfuerzo de pruebas que se requieren para producir software de alta calidad.
Realizar revisiones técnicas para valorar la estrategia de prueba y los casos de prueba	Las revisiones de prueba pueden descubrir inconsistencias, omisiones y errores evidentes en el abordaje de las pruebas.
Desarrollan un enfoque de mejora continuo para el proceso de prueba	La estrategia de prueba debe medirse. Las métricas durante las pruebas deben usarse como parte de un enfoque de control de proceso estadístico para la prueba del software.

Tabla 174.Aspectos para la realización de pruebas

4.4.4 Relación de las pruebas con otras actividades

- Pruebas vs. Pruebas de verificación y validación

  Relación de las pruebas de Verificación y Validación son una o varias pruebas de verificación complementaria y actividades de validación. Otras actividades incluyen revisiones técnicas (por ejemplo, inspecciones de código), el análisis estático, y prueba de la corrección. La especificación de una verificación completa y el proceso de validación. La verificación comprueba el funcionamiento del software, es decir que se ha implementado correctamente una funcionalidad específica. La prueba de validación comprueba si los requisitos de usuario se cumplen y los resultados obtenidos son óptimos.

- Pruebas vs depuración

  Las pruebas tienen sus intentos de provocar fallos con el fin de detectar fallos, mientras que la depuración implica tanto el análisis de fallos para localizar e identificar las fallas asociadas y corrección de fallos posteriores. Las pruebas pueden necesitar los resultados de análisis de fallos de depuración para decidir sobre un curso de acción. Estas acciones pueden incluir la terminación de la prueba o una solicitud de cambios en los requerimientos o corrección de fallo.

  Hay dos tipos de casos de prueba: Casos de prueba para la prueba, con la finalidad de los casos de prueba consiste en exponer un error no detectado previamente, y casos de prueba para la depuración, donde el propósito es proporcionar información útil en la localización de un posible error. La diferencia entre ambos es que los casos de prueba para las pruebas tienden a ser complejos porque se están tratando de cubrir muchas condiciones en un pequeño número de casos de prueba. Los casos de prueba para la depuración, por el contrario, son "escasas", ya que se desea cubrir sólo una sola condición o varias condiciones en cada caso de prueba.

En otras palabras, después de un posible error, se escriben las variantes del caso de prueba inicial para tratar de localizar el error. En realidad, este método no es un método totalmente independiente, sino que a menudo se utiliza en conjunción con el método de inducción para obtener la información necesaria para generar una hipótesis y/o para probar una hipótesis. También se utiliza con el método de deducción para eliminar las causas sospechosas, refinar la hipótesis restantes, y/o probar una hipótesis.

- Pruebas de la construcción

  Las pruebas de construcción usan planes de pruebas, diseño, casos y procedimientos desarrollados para probar las construcciones anteriores.

- Pruebas y certificación

  El enfoque de certificación involucra los siguientes aspectos:

  - Creación de escenarios de uso.
  - Especifica un perfil de uso.
  - Generación de casos de uso a partir del perfil
  - Las pruebas se ejecutan y los datos de fallo se registran y analizan
  - Se calcula la confiabilidad y se certifica

Se requiere la creación de tres modelos:

Modelo	Descripción
Modelos de muestreo	La prueba de software ejecuta n casos de prueba aleatorios y se certifica si no ocurren fallos o un número determinado de ellos. El valor n se deriva matemáticamente para afirmar que se logra la confiabilidad.
Modelo de componente	Se certifica un sistema compuesto de m componentes. Este modelo permite al analista determinar la probabilidad de que el componente fallará antes de su conclusión.
Modelo de certificación	La confiabilidad global del sistema se proyecta y se certifica.

Tabla 175.Modelos de certificación

4.4.5 Conclusiones

Destacaremos los siguientes puntos:

- El objetivo de las pruebas del software es detectar el mayor número posible de errores. Para lograrlo existen diferentes técnicas de pruebas: procedimientos técnicos o de gestión que ayudan en la ejecución, evaluación y mejora de los procesos de desarrollo software. Las técnicas

aplicadas en los procesos de prueba tienen el objetivo de seleccionar buenos casos de prueba, esto es casos que tengan una probabilidad alta de descubrir un error.

- Técnicas de caja blanca denominadas pruebas estructurales, utilizan el código fuente del programa y especialmente su estructura de control para seleccionar casos de prueba. Técnicas de caja negra o funcional, obtienen casos a partir de los requisitos funcionales del programa a probar, por lo que no se tiene en cuenta la forma en que se codifica esa funcionalidad, sino que se consideran únicamente las entradas y las salidas.

- Una característica de un sistema eficaz es la simplicidad. Un sistema sencillo puede ser comprendido más fácilmente que uno complejo.

- Los objetivos de las actividades de verificación y validación son valorar y mejorar la calidad de los productos del trabajo generados durante el desarrollo y modificación del software. Los atributos de la calidad deben ser la corrección, perfección, la consistencia, la confiabilidad, utilidad, eficacia.

- Ejecución y documentación de las pruebas de aceptación del producto final para cada producto de software.

- La visualización de los resultados ayuda de forma iterativa a guiar la búsqueda y a simplificar en gran medida tanto el análisis como la interpretación de datos.

4.4.6 Niveles de prueba

4.4.6.1 El objeto de la prueba

4.4.6.1.1 Pruebas de unidad

Las pruebas unitarias se corresponden con la prueba de cada uno de los módulos o clases del programa de forma independiente y son realizadas por el programador de forma independiente en su entorno de trabajo. En definitiva, consiste en probar los bloques más pequeños con identidad propia presentes dentro del programa. De esta forma, si una prueba descubre un nuevo error, este se encuentra más localizado. Se pueden probar a la vez varios módulos.

El enfoque de las pruebas unitarias es el de las técnicas de caja blanca. Para ello se crean módulos conductores y módulos resguardo. Un módulo conductor o módulo impulsor es un módulo específicamente para la prueba que llama al módulo a probar. Un módulo resguardo o módulo auxiliar es un módulo específicamente creado para la prueba que es llamado por el módulo a probar.

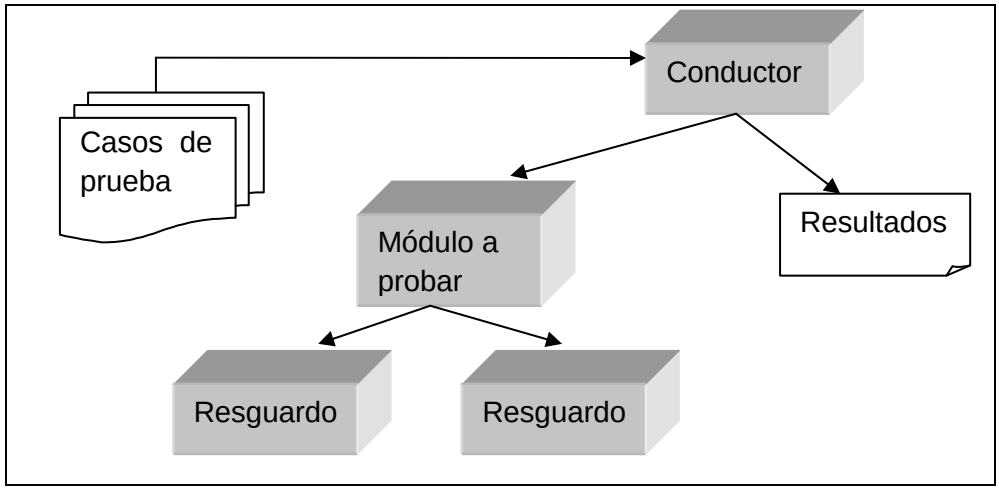


Figura 111.Pruebas unitarias. Adaptado de [Daniel Bolaños Alonso]

Se construyen módulos resguardos o módulos conductores cuya función es el paso de parámetros o variables o hacer las llamadas necesarias al módulo que se desea probar de tal forma que se pruebe el módulo de forma unitaria pero con el paso de parámetros real o desde otros módulos. De esta forma se prueba el módulo en cuestión y se corrigen los errores que surgen de dicho módulo, de tal manera que cuando se pasa a la siguiente etapa de pruebas, los módulos estén todos probados de forma independiente.

La prueba de unidad comprende el conjunto de pruebas efectuadas por un programador individual, antes de la integración de la unidad en un sistema más grande. La situación se ilustra como sigue:

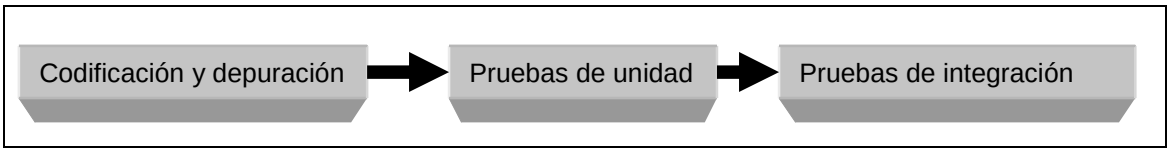


Figura 112.Pasos efectuados por un programador individual

Hay cuatro categorías de pruebas que, efectuará un programador a una unidad de prueba

Prueba	Descripción
Pruebas funcionales	Ejercitar el código con valores nominales de entrada, para los cuales se conocen los resultados esperados, además los valores límites y los valores especiales.
Pruebas de desempeño	Determinan la cantidad de tiempo de ejecución empleado en varias partes de la unidad, la eficiencia global de programa, el tiempo de respuesta, y la utilización de dispositivos por la unidad de programa.
Pruebas de tensión	Son aquellas diseñadas para romper, en forma intencional, la unidad. Están relacionadas con ejercitar la lógica interna de un programa y recorrer rutas de ejecución particulares.
Pruebas de estructura	Las actividades principales son: decidir cuales rutas ejecutar; obtener los datos de prueba para ejercitar esas rutas; determinar el criterio de cobertura de la prueba que se usará; ejecutar los casos de prueba; medir la cobertura de la prueba lograda cuando se ejercitaron esos casos.

Tabla 176.Categorías de prueba que efectuará un programador a una unidad de prueba

4.4.6.1.2 Pruebas de integración

Consiste en integrar los módulos o clases, ya probados en forma independiente en las pruebas unitarias centrándose en probar sus interfaces. Normalmente se usa el enfoque de caja negra. La cuestión principal es determinar la manera en que se combinan los distintos módulos.

Hay dos estrategias básicas:

- Prueba no incremental o big bang: En la que no hay un procedimiento establecido y se van integrando los módulos sin ningún orden establecido.

- Prueba incremental: En la que se supone un diseño en forma jerárquica o árbol y establece un orden que puede ser descendente, ascendente o sándwich. Los módulos se van integrando poco a poco con unas especificaciones establecidas.

Ventaja de los distintos tipos de integración incremental y no incremental:

- Requiere menos tiempo máquina para las pruebas, es decir que se prueba una sola vez la combinación de los módulos. Pero si contamos el tiempo de máquina para codificar todos los módulos auxiliares necesarios, la ventaja no es clara. - Existencia de probar más módulos en paralelo

Tabla 177.Ventajas de la integración no incremental

- Requiere menos trabajo, es decir se escriben menos módulos impulsores o ficticios. - Los defectos de los distintos errores se detectan en las interfaces antes, es decir, se empieza a probar uniones entre módulos. - La depuración es más fácil de realizarla es decir, si se detectan los síntomas de un defecto en un paso de la integración hay que atribuirlo al último módulo incorporado - Se prueba con más detalle el programa, al ir comprobando cada interfaz poco a poco

Tabla 178.Ventajas de la integración incremental

A continuación se detallan los tres tipos de integración incremental:

1. Pruebas de integración incremental ascendente:

   Se comienza a integrar por los módulos terminales del árbol y se continúa integrando de abajo hacia arriba hasta llegar al módulo raíz. En esta estrategia se utilizan módulos conductores y el procedimiento básico es el siguiente:

   - Se combinan módulos del nivel más bajo en grupos.
   - Se construye un conductor para coordinar la entrada y salida (E/S) de los casos de prueba.
   - Se eliminan los conductores sustituyéndolos por los módulos reales y se combinan los grupos moviéndose hacia arriba por la estructura del programa.
   - Se hacen pruebas de regresión: Repetir ciertos casos de prueba que funcionaban con el software antes de sustituir los módulos reales para asegurar que no se introducen nuevos errores.

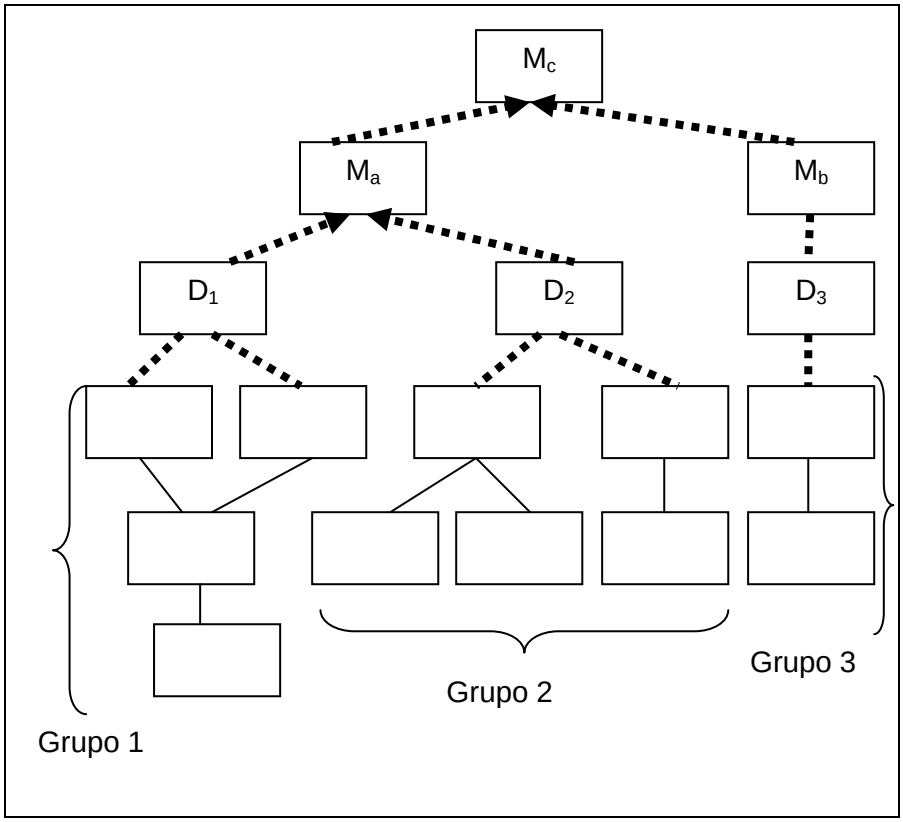


Figura 113.Prueba de integración ascendente. Adaptado de [Roger S. Pressman]

2. Pruebas de integración incremental descendente:

   Se comienza con el módulo superior y se continúa hacia abajo por la jerarquía de control bien en profundidad, bien en anchura. Se utilizan módulos resguardo. Los pasos que se siguen son:

   - Se usa el módulo control principal como conductor de pruebas, construyendo resguardos para los módulos inmediatamente subordinados.
   - Se sustituyen uno a uno los resguardos por los módulos reales.
   - Se prueba cada vez que se integra un nuevo módulo.
   - Se continúa reemplazando módulos resguardo por módulos reales hasta llegar a los nodos terminales.
   - Se hacen pruebas de regresión: Repetir ciertos casos de prueba que funcionaban con el software antes de sustituir los módulos reales para asegurar que no se introducen nuevos errores.

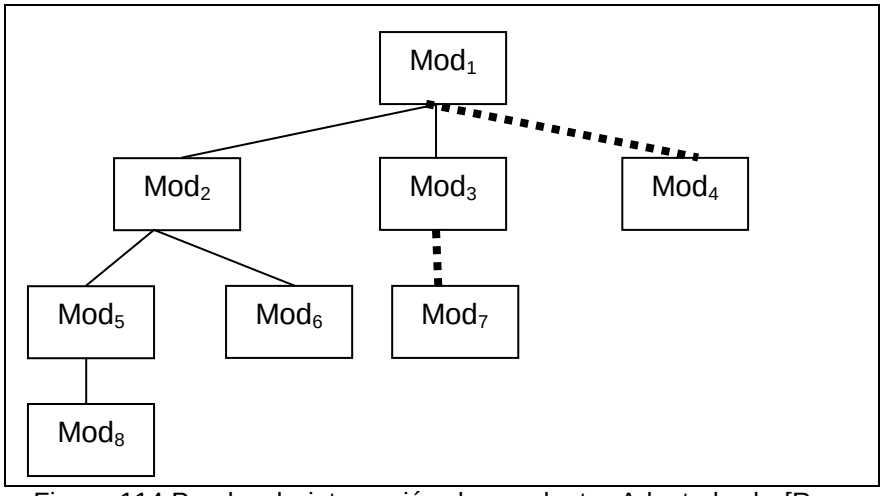


Figura 114.Prueba de integración descendente. Adaptado de [Roger S. Pressman]

3. Pruebas de integración sándwich

Esta estrategia combina las dos anteriores, es decir las aproximaciones ascendentes y descendentes. De esta forma se aplica la integración ascendente en los niveles inferiores de la jerarquía de módulos y paralelamente, se aplica la integración descendente en los niveles superiores de la jerarquía de módulos.

La integración termina cuando ambas aproximaciones se encuentran en un punto intermedio de la jerarquía de módulos.

Prueba incremental ascendente	Prueba incremental descendente
Ventajas	Ventajas
Es útil en los defectos en niveles más bajos del programa. Las entradas son fáciles de realizar Los resultados de las pruebas son más fáciles de observar	Es muy útil con grandes fallos en niveles superiores. Ingreso de funciones de entrada y salida (E/S) adquieren fácil representación de los distintos casos. Antes una estructura previa del programa.
Desventajas	Desventajas
Se requieren módulos impulsores. El programa en sí como entidad no existe hasta la incorporación del último módulo.	Difícil, la observación de los resultados. Necesarios módulos ficticios subordinados. Los módulos ficticios: es difícil su creación Incita a la superposición de diseño y pruebas

	Incita a prorrogar la terminación de la prueba en algunos módulos

Tabla 179. Comparación de ventajas e inconvenientes de las integraciones ascendentes y descendentes

4.4.6.1.3 Pruebas de sistema

La prueba del sistema es el proceso de prueba de un sistema integrado en su entorno hardware y software para verificar que cumple con los requisitos especificados, es decir:

- Cumplimiento de todos los requisitos funcionales, considerado el producto software final al completo, en un entorno del sistema.
- Funcionamiento y rendimiento en las interfaces hardware, software, usuario, operador.
- Acondicionamiento de la documentación de usuario.
- Ejecución y rendimiento en condiciones límite y de sobrecarga

En ocasiones se realiza antes que las pruebas de validación.

Entre otras pruebas consta de:

- Pruebas de interfaces externas (hardware, software, usuario).
- Prueba de volumen.
- Pruebas de funcionamiento (funciones que desempeña)
- Pruebas de recuperación.
- Pruebas de seguridad.
- Pruebas de resistencia (condiciones de sobrecarga o límite).
- Prueba de rendimiento / pruebas de comportamiento (condiciones normales).
- Pruebas de fiabilidad.
- Pruebas de documentación.

Las pruebas del sistema presentan tres fuentes principales para su diseño:

- Basado en los requisitos gracias a la técnica de caja negra aplicada a las especificaciones.
- Casos para probar el rendimiento del sistema y su capacidad funcional.
- Casos basados en el diseño de alto nivel, gracias a la técnica de caja blanca, aplicada a flujos de datos de alto nivel.

4.4.6.2 Objetivos de la prueba

4.4.6.2.1 Pruebas de aceptación/calificación

- Pruebas de aceptación

  El propósito que se desea alcanzar es comprobar si el producto está listo para ser implantado para utilización operativa en el entorno del usuario. Lo principal para evaluar el uso operativo del software suele ser la fiabilidad y la facilidad de uso del mismo. La prueba de aceptación es la prueba planificada y organizada formalmente para determinar si se cumplen los requisitos que ha determinado el cliente.

Nombre	Descripción
Participación del usuario	Por lo general es el usuario quien ejecuta las pruebas ayudado por los integrantes del equipo de pruebas. Es necesario que el mismo usuario facilite datos de prueba para la ejecución.
Prueba de los requisitos de usuario	Adoptados para este fin en forma de criterios de aceptación es decir, deben ser preferentemente objetivos y medibles. Estos criterios aseguran los requisitos del sistema, al inicio del ciclo de desarrollo.
Fase final de proceso	Crea una confianza en que el producto es el apropiado para su uso en explotación

Tabla 180.Características principales de las pruebas de aceptación

Algunas recomendaciones que pueden darse para la prueba de aceptación:

- Debe contarse con criterios de aceptación previamente aprobados por el usuario.

- Validar la documentación y los procedimientos de uso es decir, mediante auditorías.

- Las pruebas se realizarán en el entorno que se utilizará en el sistema, esto incluye al personal que lo manipula. Si se trata de un sistema contratado, las pruebas se realizan bajo inspección de la organización de desarrollo en el entorno de trabajo ayudando al usuario (prueba alfa).

  En el caso de productos de interés se entregan versiones (versiones beta) a usuarios de confianza, sin control directo, que informarán de la opinión que les merece la aplicación (pruebas beta).

- Pruebas de calificación

  El objetivo del proceso de software en las pruebas de calificación es para confirmar que el producto de software integrado cumpla con los requisitos definidos.

  Como resultado de la implementación exitosa del proceso de prueba del software de calificación:

  - Los criterios para la integración de software mediante su desarrollo es demostrar el cumplimiento de los requisitos de software.
  - El software integrado se verifica con los criterios definidos
  - Se registran los resultados de prueba
  - Se desarrolla y aplica: Una estrategia de regresión es desarrollada y aplicada para volver a probar el software integrado en el cambio de los elementos de software.

  Las actividades y tareas se pondrán en práctica de acuerdo con las políticas de la organización y la aplicación de procedimientos mediante las pruebas de calificación del software. En las pruebas de calificación para cada elemento del software (o elemento de configuración, si se ha identificado), esta actividad consiste en las siguientes tareas:

  1. El ejecutor deberá realizar las pruebas de calificación, de acuerdo con la calificación de requisitos para los elementos del software.

  2. El ejecutor deberá realizar las pruebas de calificación de acuerdo con los requisitos de calificación para el elemento software. Se deberá asegurar que la implementación de cada requisito de software es la prueba de cumplimiento. Los resultados de las pruebas de calificación deberán documentarse.

  3. El ejecutor deberá actualizar la documentación del usuario cuando sea necesario.

  4. El implementador deberá evaluar el diseño, código, pruebas, resultados de pruebas y documentación de usuario teniendo en cuenta los criterios enumerados a continuación. Los resultados de las evaluaciones deberán ser documentadas.

     a) Prueba de cobertura de los requisitos del elemento software.
     b) La conformidad con los resultados esperados.
     c) Viabilidad de la integración de sistemas y pruebas, si se realiza.
     d) Viabilidad de la operación y mantenimiento.

  5. El ejecutor será el apoyo de la auditoría (s). Los resultados de las auditorías deberán documentarse. Si el hardware y el software

están bajo desarrollo o integración, las auditorías pueden posponerse hasta las pruebas de calificación del sistema.

6. Al término de las auditorías, si se realizan, el ejecutor deberá actualizar y preparar el producto software entregable para la Integración de Sistemas, las pruebas de calificación del sistema, la instalación del software, o soporte de software de aceptación en su caso.

4.4.6.2.2 Pruebas de Instalación

Las pruebas de instalación deben ser desarrolladas por la organización que produjo el sistema, entregadas como parte del sistema, y ejecutar después de que el sistema está instalado. Entre otras cosas, los casos de prueba pueden comprobar que un conjunto compatible de opciones ha sido seleccionado, que todas las partes del sistema existen, que todos los archivos han sido creados y tienen los contenidos necesarios, y que la configuración de hardware es la adecuada.

La prueba de instalación consiste en probar la aplicación en su configuración de hardware final. Esto implica instalar la aplicación en su entorno operativo real, después de ejecutar el conjunto de pruebas del sistema. Para las aplicaciones comprimidas, las pruebas de instalación consisten en ejecutar la aplicación en plataformas que tipificarían los entornos de los clientes.

Acontecimientos que se producen en la instalación de sistemas de software. Una breve lista de ejemplos incluye lo siguiente:

- El usuario debe seleccionar una variedad de opciones.
- Los archivos y las bibliotecas deben ser asignados y cargados.
- Las configuraciones de hardware válidas deben estar presentes.
- Los programas pueden necesitar una conectividad de red para conectarse a otros programas.

4.4.6.2.3 Pruebas Alfa y Beta

Para comprobar que un producto software es realmente útil para sus usuarios es conveniente que estos últimos intervengan también en las pruebas finales del sistema. De esta manera se pueden poner de manifiesto nuevas deficiencias no caracterizadas hasta entonces. Ejemplo:

- Mensajes ininteligibles para el usuario.
- Presentación inadecuada de resultados.
- Deficiencias en el manual de usuario.
- Procedimientos de operación inusuales
- … etc.

Para la realización de estas pruebas se pueden necesitar semanas o meses porque el usuario necesita ir aprendiendo el manejo del nuevo sistema. Es

probable que los problemas más graves aparezcan ya al comienzo y por ello es aconsejable que alguien del equipo de desarrollo acompañe al usuario durante la primera toma de contacto.

Se denominan pruebas alfa a unas primeras pruebas que se realizan en un entorno controlado donde el usuario tiene el apoyo de alguna persona del equipo de desarrollo y a su vez esta misma persona puede seguir muy de cerca la evolución de las pruebas. En las pruebas beta, uno o varios usuarios trabajan con el sistema en su entorno normal, sin apoyo de nadie, y anotando cualquier problema que se presente.

En algunos sistemas pueden quedar registradas automáticamente las últimas operaciones que han dado lugar al problema. Es muy importante que el usuario sea encargado de transmitir al equipo de desarrollo cuál ha sido el procedimiento de operación que le llevó al error. Esta información es muy importante para abordar la corrección.

- Las versiones alfa: Se dan a los usuarios internos o a un grupo selecto y confiable de usuarios externos para los primeros usos antes de la liberación. Su propósito es proporcionar retroalimentación a la organización de desarrollo e información de defectos de un grupo más grande que el de los probadores, sin afectar la reputación del producto no liberado. Después de repartir las versiones alfa, se liberan las versiones beta.

- Las versiones beta: Se dan a parte de la comunidad de clientes con el entendimiento de que informarán acerca de los errores encontrados. Además, las versiones alfa y beta se usan para convencer a los clientes potenciales de que en realidad se trata de un producto que respalda las promesas de los proveedores

Usuarios internos y altamente confiables
• Multiplica las pruebas. • Pronostica la reacción de los clientes. • Beneficia a desarrolladores de terceras partes. • Anticipa la competencia.

Tabla 181.Versión alfa adaptado fuente [Eric J. Braude]

Clientes seleccionados
• Multiplica las pruebas • Obtiene la reacción del cliente

Tabla 182.Versión beta adaptado fuente [Eric J. Braude]

4.4.6.2.4 Pruebas funcionales/Pruebas de corrección

Pruebas funcionales

Las pruebas funcionales o de caja negra se centran en las especificaciones del software, del análisis de las funciones que debe realizar, de las entradas y de las salidas. A continuación nos basamos en las definiciones de Myers específicas:

- Reduce el número de otros casos necesarios para que la prueba sea razonable.
- Cubre un conjunto extenso de otros casos, es decir, nos indica algo de la ausencia o la presencia de defectos en el conjunto específico de entradas que prueba pero también de otros conjuntos similares.

Pruebas de corrección

Una vez encontrado el error debe corregirse. La corrección de un error puede introducir otros errores. Planteamiento antes de hacer la corrección que remueva la causa de un error:

- La causa del error se reproduce en otra parte del programa: En muchas situaciones, un defecto de programa es causado por un patrón de lógica erróneo que puede reproducirse en alguna otra parte. La consideración del patrón lógico puede resultar en el descubrimiento de otros errores.

- Qué "siguiente error" puede introducirse con la corrección que está a punto de realizar: antes de hacer la corrección, debe evaluar el código fuente o tal vez el diseño, para valorar el acoplamiento de las estructuras lógicas y de datos. Si la corrección se realizará en una sección altamente acoplada del programa, debe tenerse especial cuidado cuando se realice algún cambio.

- Qué debió hacerse para evitar este error desde el principio: Este es el primer paso hacia el establecimiento de un enfoque de aseguramiento de la calidad estadística del software. Si se corrigen tanto el proceso como el producto, el error se removerá del programa actual y podrá eliminarse de todos los programas futuros.

4.4.6.2.5 Materialización de la confiabilidad y evaluación

Los argumentos de confiabilidad son documentos estructurados que proporcionan argumentos detallados y evidencias de que un sistema es seguro o de que se ha alcanzado un nivel requerido de confiabilidad. La confiabilidad de un sistema es una propiedad del sistema que es igual a su fidelidad significa el grado de confianza del usuario en que el sistema operará tal y como se espera de él, y no fallará al emplearlo.

Esta propiedad no se puede expresar numéricamente, sino que se utilizarán términos como: muy confiable, no confiable, ultraconfiable, estos términos reflejan los grados de confianza que se podrían tener en cualquier sistema.

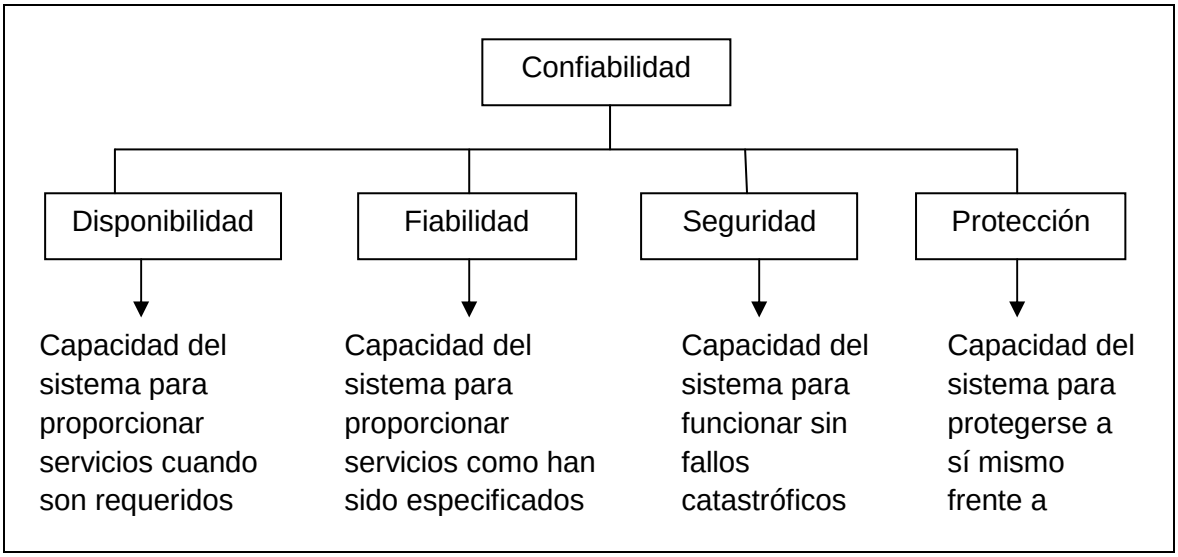


Figura 115.Dimensiones de la confiabilidad. Adaptado de [Ian Sommerville]

Dimensiones	Descripción
Disponibilidad	Probabilidad de que esté activo y en funcionamiento para otorgar servicios rentables
Fiabilidad	Probabilidad de que durante un determinado periodo de tiempo, el sistema funcione tal y como espera el usuario
Seguridad	Valoración de la probabilidad de que el sistema ocasione daño tanto al sistema como a su entorno
Protección	Valoración de la probabilidad de que el sistema pueda resistir intrusiones accidentales o premeditadas

Tabla 183.Descripción detalla de la figura 115

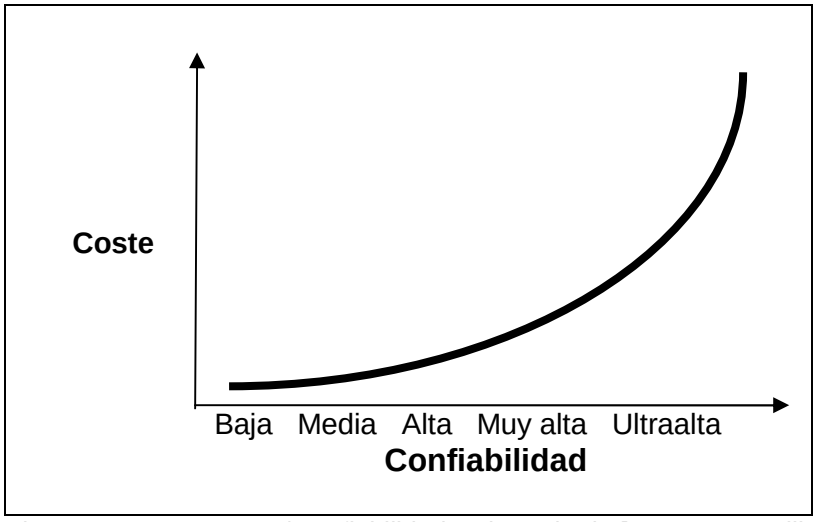


Figura 116.Curva coste/ confiabilidad. Adaptado de [Ian Sommerville]

A continuación explicaremos la composición en otras propiedades:

- La protección incluye integridad, asegurar que el programa y los datos de los sistemas no resulten dañados.
- Confidencialidad, asegurar que las personas autorizadas puedan acceder a la información
- Fiabilidad incluye la corrección, asegurar que los servicios que proporciona al sistema son los que se han especificado.
- Precisión, asegurar que la información es proporcionada al usuario con el nivel de detalle adecuado.
- Oportunidad, asegurar que la información que proporciona el sistema se hace cuando es requerida.

Se pueden obtener otras propiedades del sistema incluidas en términos de confiablidad.

Nombre	Descripción
Reparabilidad	Los fallos de funcionamiento del sistema son inevitables. La reparabilidad se mejora cuando se tiene acceso al código fuente.
Mantenibilidad	A medida que se utiliza el sistema, se manifiestan nuevos requisitos. Un software mantenible, es un software que puede adaptarse para tener en cuenta los nuevos requisitos con un coste razonable y con una baja probabilidad de ingresar nuevos errores al sistema.
Supervivencia	Característica primordial en los sistemas basados en internet es la supervivencia, está relacionada con la seguridad y disponibilidad. La supervivencia es la capacidad de un sistema para continuar ofreciendo su servicio mientras está siendo atacado y, potencialmente parte del sistema está inhabilitado
Tolerancia a fallos	Puede considerarse como parte de la usabilidad, refleja hasta qué punto el sistema ha sido diseñado para evitar y tolerar un error en la entrada de datos del usuario al sistema.

Tabla 184.Propiedades del sistema incluidas en el término de confiabilidad

4.4.6.2.6 Pruebas de Regresión

Cuando se realiza un cambio en el software, es necesario volver a probar comportamientos y aspectos ya probados de forma satisfactoria anteriormente y volver a utilizar algunos de los conjuntos de los casos de prueba diseñados para asegurar que el cambio efectuado en el software no ha inferido en el correcto comportamiento del sistema software y no se han producido efectos colaterales. Este tipo de pruebas son, por tanto muy comunes durante la fase de mantenimiento de un sistema operativo.

Cuando una aplicación crece, las pruebas adquieren un significado especial. Esto es en especial notorio cuando se hace un cambio en una aplicación grande y los desarrolladores necesitan validar el hecho de que el cambio no

haya afectado a la funcionalidad existente. Las pruebas de regresión se realizan a menudo. Si el tiempo no permite no realizar las pruebas completas, entonces se seleccionan aquellas con mayor probabilidad de fallar debido a los cambios.

4.4.6.2.7 Pruebas de Rendimiento

Sirven para comprobar las prestaciones del sistema que son críticas en el tiempo. Estas pruebas son fundamentales para los sistemas de tiempo real o sistemas embebidos. Para medir los tiempos se necesitan equipos de instrumentación. Ejemplo, se puede forzar al sistema provocando múltiples interrupciones simultáneamente para medir el tiempo máximo que se emplea en tratarlas.

La prueba de rendimiento ocurre a lo largo de todos los pasos de prueba e incluso en el nivel de unidad, se puede acceder al rendimiento de un módulo individual a medida que se desarrollan las pruebas.

Las pruebas de rendimiento con normalidad se acoplan con las pruebas de esfuerzo. Las pruebas de esfuerzo se diseñan para enfrentar los programas con situaciones anormales; las pruebas de esfuerzo ejecutan un sistema en forma que demanda recursos en cantidad, frecuencia o volumen anormales. Y por tanto la prueba de rendimiento como la prueba de esfuerzo requieren instrumentación de hardware y software, es decir a menudo es necesario medir la utilización de los recursos.

En la instrumentación externa se puede monitorear a intervalos de ejecución y eventos de registros. En la instrumentación de un sistema las personas que realicen las pruebas pueden descubrir situaciones que conduzcan a la degradación y posibles fallos del sistema.

Las pruebas de rendimiento tienen que diseñarse para asegurar que el sistema pueda procesar su carga esperada. Esto implica planificar una serie de pruebas en las que la carga se ve incrementada regularmente hasta que el rendimiento del sistema se hace inaceptable. Además las pruebas de rendimiento se ocupan de demostrar que el sistema satisface los requisitos como de descubrir problemas y defectos en el sistema.

4.4.6.2.8 Pruebas de Recuperación

Programas tales como sistemas operativos, sistemas de gestión de datos, etc. tienen objetivos de recuperación de ese estado en el funcionamiento del sistema en la recuperación de los errores de programación, errores de hardware y los errores de datos.

Uno de los objetivos de la prueba del sistema es mostrar que la recuperación de estas funciones no funciona correctamente. Los errores de programación

pueden ser intencionalmente inyectados en un sistema para determinar si puede recuperarse de ellos. Los fallos de hardware, tales como errores de paridad de memoria o un dispositivo de errores de entrada y salida (E/S) se pueden simular. Los errores de datos o un puntero no válido en una base de datos se pueden crear a propósito o simular para analizar la reacción del sistema.

Uno de los objetivos del diseño de estos sistemas es minimizar el tiempo medio de recuperación (MTTR). El tiempo de inactividad a menudo causa que una empresa pierda ingresos porque el sistema no funciona. Uno de los objetivos de prueba es mostrar que el sistema no cumple con el acuerdo de nivel de servicio de tiempo medio de recuperación. A menudo, el tiempo medio de recuperación tendrá un límite superior e inferior, por lo que sus casos de prueba deben reflejar estos límites.

4.4.6.2.9 Pruebas de Configuración

Programas tales como sistemas operativos, sistemas de gestión de datos y programas de conmutación de mensajes de apoyo de gran variedad de hardware de configuraciones, incluyendo diferentes tipos y números de E/S y dispositivos de líneas de comunicación, o de distintos tamaños de memoria.

A menudo, el número de configuraciones posibles es demasiado grande para poner a prueba cada una de ellas, pero por lo menos, debe probar el programa con cada tipo de dispositivo de hardware y con la configuración mínima y máxima. Si el propio programa puede ser configurado para omitir los componentes del programa, o si el programa puede ejecutarse en equipos diferentes, de cada posible configuración del programa debe hacerse la prueba.

Hoy en día, muchos programas están diseñados para múltiples sistemas operativos, por ejemplo, que si se está probando un programa, debe probarlo con todos los sistemas operativos para los cuales fue diseñado. Los programas diseñados para ejecutarse en un navegador de Internet requieren una atención especial, ya que existen numerosos navegadores web disponibles y no todos funcionan del mismo modo.

El software debe ejecutarse en varias plataformas y bajo más de un entorno de sistema operativo. La prueba de configuración, ejercita el software en cada entorno en que debe operar; examina los procedimientos de instalación y el software de instalación especializado que usarán los clientes, así como toda la documentación que se usará para introducir el software a los usuarios finales.

4.4.6.2.10 Pruebas de Facilidad de Uso

Las pruebas de uso se diseñan para determinar el grado de la interfaz que facilita la vida del usuario. Las pruebas finales las realizará el usuario final siguiendo para ello la siguiente secuencia de pasos:

- Definir un conjunto de prueba de uso mediante categorías e identificar las metas que se persiguen de cada una.
- Diseño de pruebas que conllevan la evaluación de cada una de las metas que se deben alcanzar.
- Seleccionar al usuario que desarrollará las pruebas.
- Realizar un mecanismo para valorar el uso

Las pruebas de uso se representan en diferentes niveles de abstracción:

- Valorar el uso mediante un mecanismo de interfaz específico.
- Evaluar el uso de una página web es decir que contemple mecanismos de objetos de datos, interfaz, funciones que estén relacionadas.
- Considerar el uso

4.4.6.2.11 Desarrollo dirigido por Pruebas

El enfoque de cada proceso de prueba, se centra en una determinada clase de errores. El ciclo de prueba se ha estructurado en el modelo de ciclo de desarrollo. En otras palabras, debe ser capaz de establecer una correspondencia uno-a-uno entre el desarrollo y prueba. Por ejemplo:

- El propósito de una prueba de módulo es el de encontrar discrepancias entre los módulos del programa y sus especificaciones de la interfaz.
- El propósito de una prueba de la salida es demostrar que un programa no coincide con sus especificaciones externas.
- El propósito de una prueba del sistema consiste en demostrar que el producto es incompatible con sus objetivos originales.

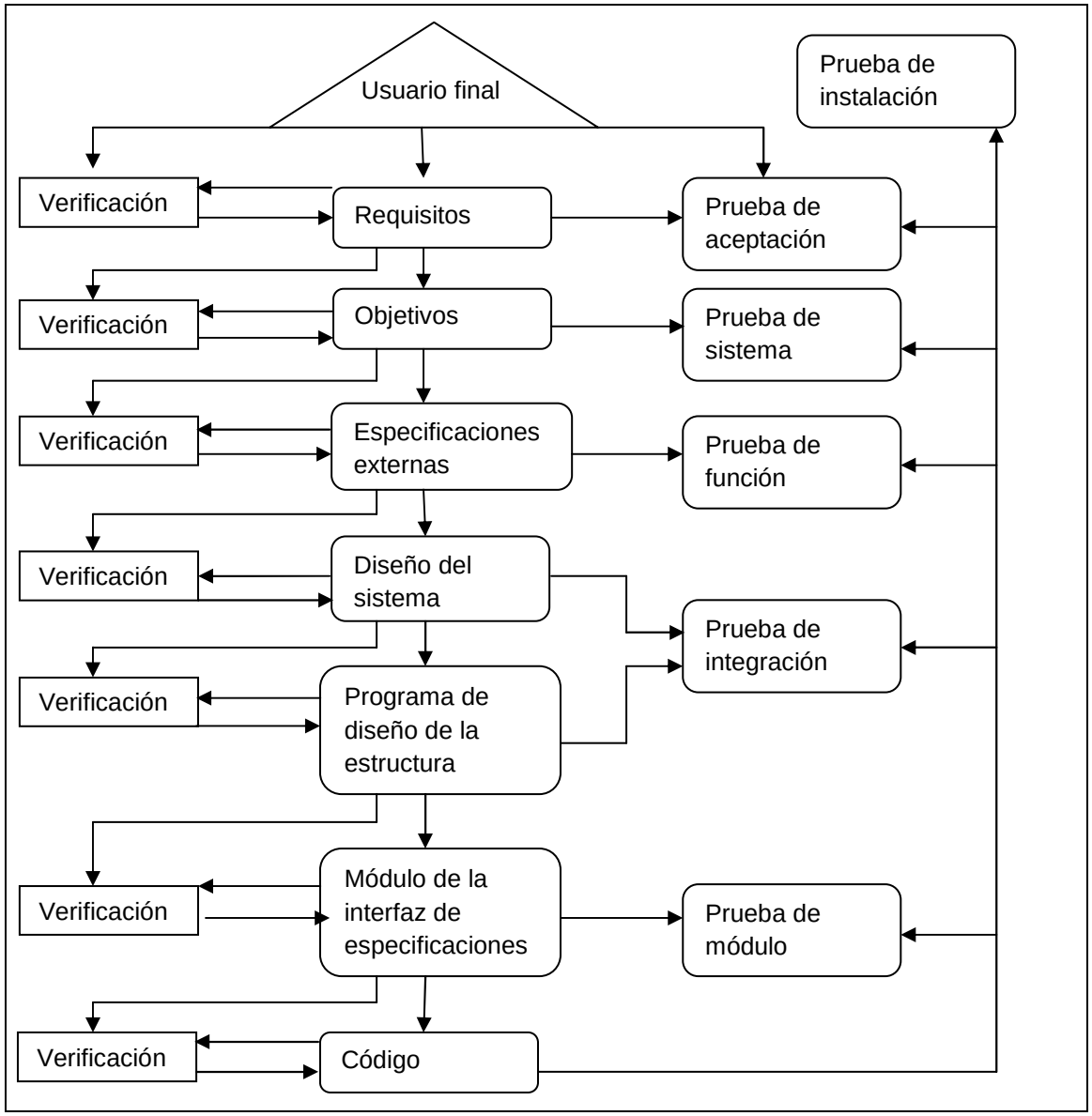


Figura 117.Correspondencia entre los procesos de desarrollo y pruebas. Adaptado de [Glenford J. Myers]

Las ventajas de esta estructura es evitar las pruebas redundantes e impedir clases con vistas a gran cantidad de errores. Las realizadas durante la traducción de los objetivos de la especificación externa, se mide con respecto a un tipo distinto de documentación en el proceso de desarrollo.

Los métodos de análisis de orden superior son más aplicables a los productos de software es decir, programas escritos como resultado de un contrato o los programas destinados para uso general, a diferencia de los programas experimentales o los programas escritos para ser utilizados exclusivamente por el autor del programa.

Los programas no están escritos como productos a menudo no tienen requisitos formales y sus correspondientes objetivos, porque en este tipo de programas, la prueba de función podría ser la única prueba de orden superior. Las pruebas de función son un proceso de tratar de encontrar discrepancias entre el programa y la especificación externa.

Una especificación externa es una descripción precisa del comportamiento del programa desde el punto de vista del usuario final. Excepto cuando se usa en pequeños programas, las pruebas de función son normalmente una actividad de recuadro negro, es decir, se basan en el anterior módulo de proceso de pruebas para lograr la deseada caja blanca de cobertura lógica de los criterios.

Para llevar a cabo una prueba de función, la especificación se analiza para obtener un conjunto de casos de prueba. La partición de equivalencia, los métodos de análisis de valores en la frontera, gráficos de causa-efecto, son especialmente pertinentes para las pruebas de función.

Además, la necesidad de pruebas de orden superior aumenta a medida que crece el programa. La razón es la proporción de errores en el diseño (errores en los procesos de desarrollo anterior) a los errores de codificación es considerablemente mayor en los programas grandes que en los pequeños programas.

Se debe tener en cuenta que la secuencia de las pruebas de los procesos en la figura 117, no implica necesariamente una secuencia de tiempo. Por ejemplo, ya que las pruebas del sistema no se han definido como "el tipo de pruebas que se hacen después de las pruebas de función", sino que se define como un tipo distinto de pruebas centradas en una clase distinta de errores.

Las pruebas de integración se omiten debido a que a menudo no son consideradas como un paso de prueba por separado y, cuando las pruebas del módulo incremental se utilizan, es una parte implícita de la prueba del módulo. El análisis de estos procesos de prueba será breve, en general, y, en su mayor parte, sin ejemplos ya que las técnicas específicas utilizadas en estas pruebas de orden superior son muy dependientes del programa específico que se está probando.

Por ejemplo, las características de una prueba del sistema (los tipos de casos de prueba, la manera en que los casos de prueba se diseñan, las herramientas de prueba que se utilizan, etc.) para un sistema operativo difieren considerablemente de un sistema de prueba de un compilador, o un programa de base de datos de aplicación.

4.4.7 Conclusiones

Destacaremos los siguientes puntos:

- El objetivo de cada uno de los niveles de pruebas se realiza siguiendo un plan de desarrollo que comienza con la implementación de componentes individuales con interfaces bien definidas. A estos módulos se les denomina pruebas unitarias. Al integrar los distintos componentes pueden seguir errores e interacciones que son el objetivo de las pruebas de integración. Sobre el software ya completo se realizan pruebas del sistema, y tras estas se instala en su entorno de operación.

- Los procesos de pruebas no se deben plantear en las fases finales de la implementación del producto, como se gestionaba en los modelos tradicionales de desarrollo del software: el retraso en la detección de errores aumenta el coste derivado de su corrección.

- La utilidad de un sistema se refiere a la facilidad de uso. La facilidad de uso siempre debe estar presente en todas las etapas del ciclo de vida del software.

- La corrección permite que los productos software realicen adecuadamente las tareas definidas por su especificación.

- La usabilidad es un factor importante que distingue la calidad de un producto software.

4.4.8 Técnicas de prueba

4.4.8.1 Pruebas Basadas en la Intuición y Experiencia

4.4.8.1.1 Pruebas ad hoc

Las pruebas ad hoc también son conocidas como pruebas de valor especial es probablemente la forma más practicada de las pruebas funcionales. También es el más intuitivo y menos uniforme. Prueba de valor especial se produce cuando un probador utiliza su conocimiento del dominio, la experiencia con programas similares, y la información para diseñar casos de prueba. Como resultado, las pruebas de valor especial son muy dependientes de las habilidades del probador.

A pesar de todos los aspectos negativos evidentes en especial para la función NextDate, se encuentra que no es muy satisfactorio. Si un probador define en

especial los casos de prueba de valor para NextDate, veríamos varios casos de prueba que implican el 28 de febrero, 29 de febrero, y los años bisiestos.

Pero la prueba de valor especial puede ser muy útil y además muy subjetiva, a menudo resulta en un conjunto de casos de prueba ser más eficaz en los errores que revelan prueba de conjuntos generados que otros métodos que hemos estudiado en las pruebas de software.

4.4.8.2 Basadas en la Especificación

4.4.8.2.1 Particiones de Equivalencia

Ya que es imposible realizar pruebas que contemplan todos los casos y caminos posibles el propósito de esta técnica es cubrir las pruebas de la forma más amplia diseñando y usando un número de casos de prueba manejable. El objetivo en la partición de equivalencia es dividir la entrada de cualquier programa en clase de datos que se puedan derivar para reducir los casos de prueba.

Para ello se obtiene cada valor de entrada y se divide en varios grupos, y para ello se contemplan los siguientes aspectos:

- El valor de entrada es un rango, se establece una clase de equivalencia válida y dos inválidas.
- El valor de entrada es un valor específico, se establece una clase de equivalencia válida y dos inválidas.
- El valor de entrada es un valor lógico, se establece una clase de equivalencia válida y dos inválidas.

Las clases de equivalencia forman una partición de un conjunto, donde la partición se refiere a una colección de subconjuntos mutuamente disjuntos cuya unión es todo el conjunto. Esto tiene dos implicaciones importantes para la prueba: el hecho de que todo el conjunto está representado proporciona una forma de integridad, y la disyunción asegura una forma de no redundancia. Debido a que los subconjuntos son determinados por una relación de equivalencia, los elementos de un subconjunto tienen algo en común.

La idea de las pruebas de clase de equivalencia es la identificación de casos de prueba mediante el uso de un elemento de cada clase de equivalencia. Si las clases de equivalencia son elegidas adecuadamente se reduce la redundancia entre los casos de prueba. Como ejemplo tomaremos un programa cuya función (F) tiene tres variables a, b, c, y cuyo dominio de entrada se compone de los conjuntos A, B y C. Suponemos que elegimos adecuadamente relación de equivalencia, lo que induce a la partición siguiente:

$$A = A_I \cup A_2 \cup A_3$$
$$B = B_I \cup B_2 \cup B_3 \cup B_4$$
$$C = C_I \cup C_2$$

Figura 118.Elección adecuada de las clases de equivalencias cuyo dominio de entrada se compone de los conjuntos A,B, C

$$a1 \in A1$$
$$b3 \in B3$$
$$c2 \in C2$$

Figura 119.Elementos de las particiones

A continuación se distingue entre clase de equivalencia débil y clase de equivalencia fuerte:

Prueba de clase de equivalencia débil	Prueba de clase de equivalencia fuerte
Con la anotación que se encuentra en las figuras 118, 119. La prueba de clase de equivalencia débil “a” se logra mediante el uso de una variable de cada clase de equivalencia en un caso de prueba.	Prueba de clase de equivalencia fuerte se basa en el producto cartesiano de los subconjuntos de la partición. El producto cartesiano garantiza que tenemos una noción de "integridad" en dos sentidos: cubrimos todas las clases de equivalencia, y tenemos uno de cada posible combinación de entradas.

Tabla 185.Diferencia entre clase de equivalencia débil y fuerte

Caso de prueba	a	b	c
clase de equivalencia débil 1	a_1	b_1	c_1
clase de equivalencia débil 2	a_2	b_2	c_2
clase de equivalencia débil 3	a_3	b_3	c_3
clase de equivalencia débil 4	A_4	b_4	c_4

Tabla 186.Ejemplo de casos de prueba de clase de equivalencia débil

Este conjunto de casos de prueba utiliza un valor de cada clase de equivalencia. Identificamos estos de una manera sistemática. De hecho, siempre tendremos el mismo número de casos de Prueba de clase de equivalencia débil, ya que hay clases en la partición con el mayor número de subconjuntos.

Caso de prueba	a	b	c
clase de equivalencia fuerte 1	a_1	b_1	c_1
clase de equivalencia fuerte 2	a_1	b_1	c_2
clase de equivalencia fuerte 3	a_1	b_2	c_1
clase de equivalencia fuerte 4	a_1	b_2	c_2
clase de equivalencia fuerte 5	a_1	b_3	c_1
clase de equivalencia fuerte 6	a_1	b_3	c_2
clase de equivalencia fuerte 7	a_1	b_4	c_1

clase de equivalencia fuerte 8	a_1	b_4	c_2
clase de equivalencia fuerte 9	a_2	b_1	c_1
clase de equivalencia fuerte 10	a_2	b_1	c_2
clase de equivalencia fuerte 11	a_2	b_2	c_1
clase de equivalencia fuerte 12	a_2	b_2	c_2
clase de equivalencia fuerte 13	a_2	b_3	c_1
clase de equivalencia fuerte 14	a_2	b_3	c_2
clase de equivalencia fuerte 15	a_2	b_4	c_1
clase de equivalencia fuerte 16	a_2	b_4	c_2
clase de equivalencia fuerte 17	a_3	b_1	c_1
clase de equivalencia fuerte 18	a_3	b_1	c_2
clase de equivalencia fuerte 19	a_3	b_2	c_1
clase de equivalencia fuerte 20	a_3	b_2	c_2
clase de equivalencia fuerte 21	a_3	b_3	c_1
clase de equivalencia fuerte 22	a_3	b_3	c_2
clase de equivalencia fuerte 23	a_3	b_4	c_1
clase de equivalencia fuerte 24	a_3	b_4	c_2

Tabla 187.Ejemplo de casos de prueba de clase de equivalencia fuerte

El producto cartesiano A x B x C tendrá 3 x 4 x 2 = 24 elementos, resultado de los casos de prueba en la tabla 187. La similitud entre el patrón de estos casos de prueba y la construcción de una tabla de verdad en la lógica proposicional. El producto cartesiano garantiza que tenemos una noción de integridad en dos sentidos:

- Cubrimos todas las clases de equivalencia, y tenemos uno de cada posible combinación de entradas. La mayoría de las veces, las pruebas de clase de equivalencia definen las clases del dominio de entrada.
- Definir relaciones de equivalencia en el rango de salida de la función del programa que se pone a prueba.

4.4.8.2.2 Análisis de los Valores Límite

El análisis de los valores límites se centra en el límite del espacio de entrada para identificar casos de prueba. El fundamento de las pruebas de valor de límite es que los errores tienden a ocurrir cerca de los valores límites de una variable de entrada.

La idea básica del análisis de valores límite es el uso de valores de entrada, es obtener un valor nominal mínimo y valor nominal máximo para controlar el valor de las variables fronteras. La siguiente parte del análisis de valor límite se basa en un supuesto crítico, es conocido como "único fallo" hipótesis de la teoría de la fiabilidad, este supuesto dice que los fallos son rara vez el resultado de la ocurrencia simultánea de dos o más errores.

El límite del valor de los casos de prueba de análisis de una función F de dos variables son las siguientes:

{<X_{1nom}, X_{2min}>, <X_{1nom}, $X_{2min+>,}$ <X_{1nom},X_{2nom}>, <X_{1nom}, X_{2max-}> <X_{1nom}, X_{2max}>, <X_{1min}, X_{2nom}>, > $X_{1min+,}X_{2nom}$>, <X_{1nom}, X_{2nom}> <X_{1max-}, X_{2nom}>, <X_{1max}, $X2_{nom}$> }

Figura 120.Variables cuyo valor se controla en los valores límite

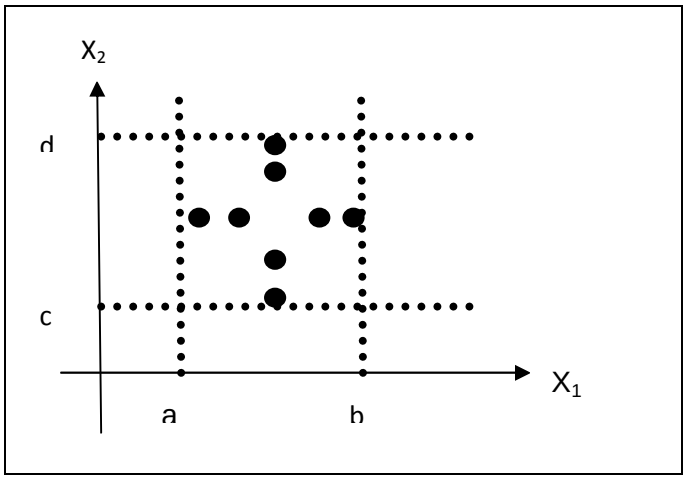


Figura 121.Casos de prueba de análisis de los valores en los límites para una función F de dos variables. Adaptado de [Paul C. Jorgensen]

El valor básico de los límites de análisis se puede generalizar de dos maneras:

Nombre	Descripción
Número de variables	La generalización del número de variables es fácil: si tenemos una función de n variables, tenemos todos menos uno en el valor nominal, y las variables restantes asumen valores como pueden ser: min, min +, nom, max-y max; repetir esta operación para cada variable. Para una función de n variables, el análisis del valor límite del rendimiento es de 4n + 1.
Tipo de rango	La generalización de los rangos depende de la naturaleza o más precisamente del tipo de las propias variables. En la función NextDate, por ejemplo, tenemos las variables para el mes, el día, y el año. En un lenguaje FORTRAN se codifican de la siguiente manera para el mes de enero se asigna el número 1, para el mes de febrero el mes 2 y así sucesivamente. En el lenguaje Pascal se admiten los tipos definidos por el usuario la variable mes como un tipo enumerado { enero, febrero….diciembre}

Tabla 188.Generalización del valor de los límites

4.4.8.3 Tablas de Decisión

Las tablas de decisión se han utilizado para representar y analizar complejas relaciones lógicas. Son ideales para describir situaciones en las que se toman una serie de combinaciones de acciones en diversas series de condiciones.

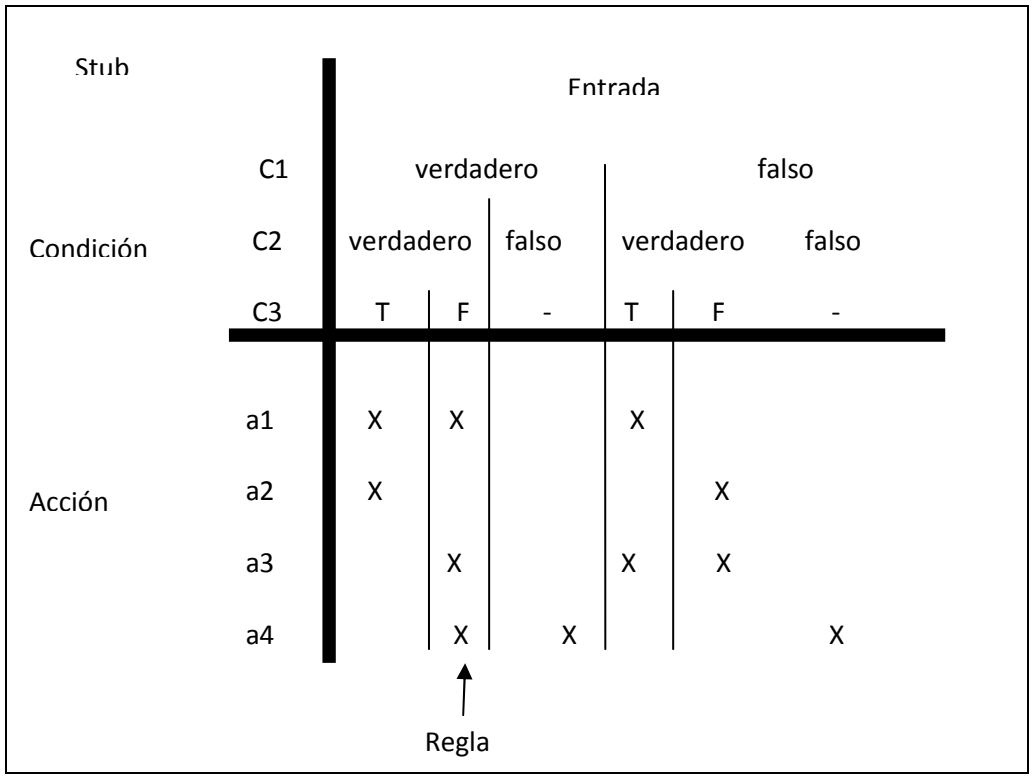


Figura 122.Tabla de decisión. Adaptado de [Paul C. Jorgensen]

Partes en la tabla de decisión:

- La derecha es la parte de entrada.
- La parte de arriba de la línea gruesa es la parte de condición.
- La parte de abajo de la línea gruesa es la parte de la acción.

Una columna en la parte de entrada es una regla.

Cuando tenemos condiciones binarias (verdadero / falso, sí/no, 0/1), la parte de la condición de una tabla de decisión es una tabla de verdad lógica proposicional. Esta estructura garantiza que consideramos todas las combinaciones posibles de valores de condición.

Cuando utilizamos las tablas de decisión para la identificación de casos de prueba, la propiedad de integridad de una tabla de decisión garantiza pruebas completas. Las tablas de decisión en las que todas las condiciones son binarias se llaman tablas de decisión de entradas limitadas. Si las condiciones permiten tener varios valores, las tablas resultantes se denominan tablas de decisión de entrada extendida.

Para identificar los casos de prueba con tablas de decisión, interpretamos las condiciones como entradas y acciones como salidas. A veces, las condiciones

terminan refiriéndose a las clases de equivalencia de las entradas, y las acciones se refieren a las principales partes funcionales de procesamiento del elemento que se prueba. Las reglas son interpretadas como casos de prueba. Debido a que la tabla de decisiones mecánicamente puede ser obligada a ser completa, sabemos que tenemos un conjunto completo de casos de prueba.
Existen varias técnicas que producen tablas de decisión que son útiles a los probadores.

4.4.8.4 Basadas en Máquinas de Estado Finito

Las máquinas de estado finito se han convertido en una notación bastante estándar para la especificación de requisitos. Todas las extensiones de tiempo real de análisis estructurado usan alguna forma de máquina de estados finitos, y casi todas las formas de análisis orientado a objetos así lo requieren.

Las máquinas de estados finitos pueden ser ejecutadas, pero en una serie de convenciones se necesitan en primer lugar. Una es la noción del estado activo. Hablamos de un sistema que está "en" un cierto estado, cuando el sistema se modela como una máquina de estados finitos, el estado activo se refiere al estado "en que estamos".

De manera más formal:

Una máquina de estados finitos (s un grafo dirigido (S, T, Ev, Act), en la que S es un conjunto de nodos (los estados), T es un conjunto de aristas (transiciones), y Ev y Act establecen eventos y acciones que se asocian con las transiciones en T.

Las máquinas de estados finitos pueden ser ejecutadas, pero en una serie de convenciones se necesitan en primer lugar. Una es la noción del estado activo. Hablamos de un sistema que se está "en" un cierto estado, cuando el sistema se modela como una máquina de estados finitos, el estado activo se refiere al estado "en que estamos".

Otra noción, es que las máquinas de estados finitos pueden tener un estado inicial, que es el estado que se activa cuando una máquina de estados finitos entró por primera vez. (El estado de reposo es el estado inicial, lo que es indicado por la transición que viene de ninguna parte final de los estados, se reconocen por la ausencia de transiciones de salida...).

Exactamente un estado puede ser activo en cualquier momento. Pensamos también en las transiciones como ocurrencias instantáneas, y los eventos que causan las transiciones también se producen de uno en uno. Para ejecutar una máquina de estados finitos, se comienza con un estado inicial, y proporcionan una secuencia de eventos que hacen que las transiciones de estado se ejecuten. A medida que aparece cada evento, la transición cambia al estado activo, y se produce un nuevo evento. De esta manera, una secuencia de eventos selecciona una ruta de los estados (o equivalente, de las transiciones) a través de la máquina.

4.4.8.5 Basadas en las especificaciones formales

Decir que un programa tiene demostración formal de que es correcto significa que proporciona una demostración matemática para enseñar que el programa satisface sus requisitos. La demostración se basa en los requisitos para el código y el texto del código. La demostración es independiente de la compilación y es independiente de las pruebas. Esta es una capacidad ideal.

Una buena demostración matemática es un argumento bien construido que convence y que la declaración es cierta. El lenguaje de matemáticas es la clave de las demostraciones, pero estas pueden aplicarse solo para demostrar declaraciones expresadas en términos matemáticos.

Las matemáticas son buenas para expresar el estado en otras palabras *lo que es*. Esto difiere de los procedimientos y algoritmos para el *cómo.* Dado que las especificaciones de los requisitos en su mayor parte describen el estado de la aplicación antes y después de los eventos, la notación matemática puede ser más adecuada en el lenguaje natural para especificar los requisitos detallados.

El uso de las matemáticas en este contexto es parte de lo que denominamos *métodos formales. C*omo ejemplo consideramos la siguiente especificación para un procedimiento:

Devuelve una matriz ordenada con los elementos de la matriz B.

Este requisito aparentemente sencillo tiene un sorprendente contenido de ambigüedad. Los elementos de la matriz B se pueden ordenar de varias formas como puede ser: ordenado por pares, impares, creciente, decreciente por tanto el requisito sigue siendo ambiguo.

Una notación común para expresar los requisitos de manera formal se conoce como especificaciones “Z”. Esta es la forma estándar de describir el estado requerido antes y después de un procedimiento.

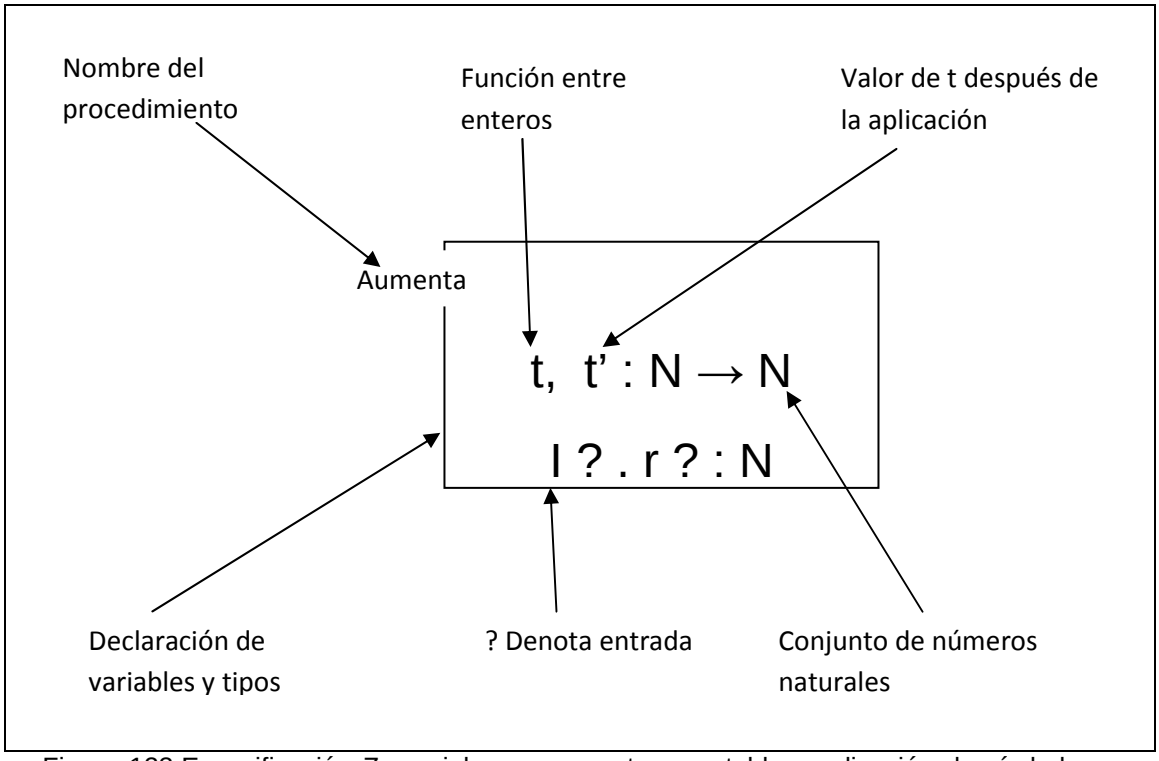


Figura 123.Especificación Z parcial para aumentar una tabla: explicación de símbolos. Adaptado de [Eric J. Braude]

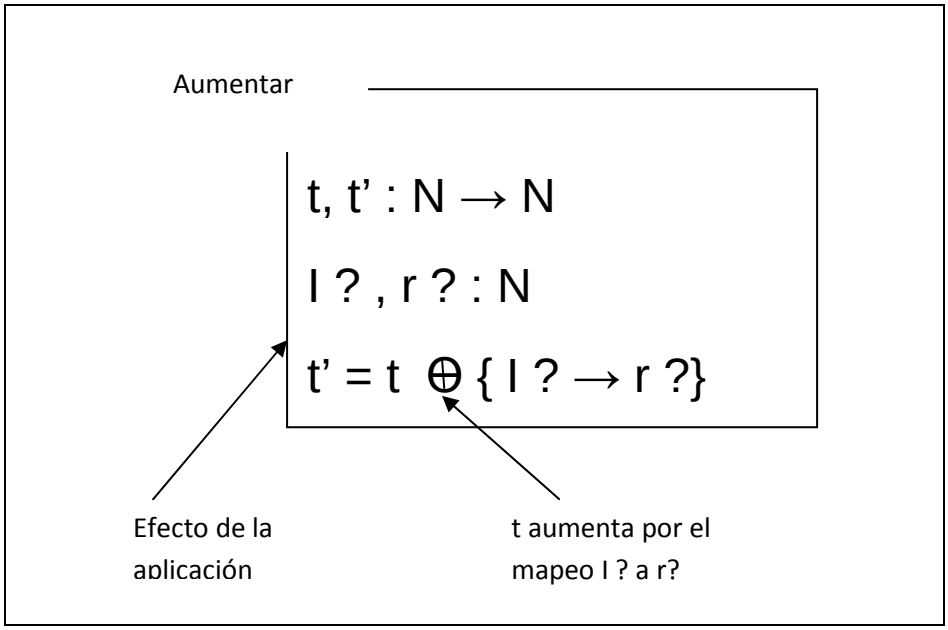


Figura 124.Especificación Z completa para aumentar una tabla. Adaptado de [Eric J. Braude]

¿Cuándo se debe utilizar la especificación formal?

La efectividad al comunicar los requisitos es la mejor prueba para utilizar requisitos formales e informales. Con frecuencia es mejor expresar los requisitos en lenguaje natural y algunas veces es preferible hacerlo mediante una especificación formal. Si el requisito puede explicarse en términos de una

salida concreta y los desarrolladores tienen conocimiento de los métodos formales, entonces los requisitos formales serán un medio factible para expresar los requisitos específicos.

Las especificaciones formales se requieren para implementar las especificaciones ejecutables. Estas especificaciones pueden traducirse de manera automática en código objeto y por lo tanto deben ser precisas.

Precondiciones y poscondiciones

Estas son descripciones del estado requerido de la aplicación antes y después de un cálculo. Las precondiciones y poscondiciones usan seudocódigo al igual que partes del lenguaje de implementación como java, C++, etc., en lugar de sólo matemáticas. Por ello se aplica con frecuencia para diseños específicos.

Precondición

P es un matriz de longitud long
0 < long < 1000

Poscondición

0 ≤ n < long // n es índice máximo
m = P[n] // m es el valor máximo
P[i] ≤ m para i=0… long -1
Si P[j]= m entonces j ≥ n // primero

Figura 125.Especificaciones Z contra precondiciones y poscondiciones

4.4.8.5.1 Pruebas aleatorias

Se usan entradas de pruebas aleatorias para evitar muestras sesgadas en las pruebas. Se selecciona el tipo de prueba como cobertura de decisiones y definidos los límites de los datos, todavía queda una libertad sustancial para las entradas posibles. En el caso ideal, esta entrada debe elegirse al azar.

Las pruebas aleatorias simulan la entrada del programa creando datos que luego se introducirán para que realicen la secuencia y la frecuencia con la que se podrán utilizar en la práctica del día a día, de forma continua. Esto conlleva utilizar herramientas generadoras de pruebas que se nutren con la descripción de datos y secuencias de entradas así como una estimación de probabilidad que pueda ocurrir con cada una de ellas al utilizarlas en el uso real. Este enfoque es habitual en la prueba de compiladores que generan programas codificados aleatoriamente en un lenguaje concreto que sirve de pruebas para verificar si son compilados correctamente.

El proceso de generación de pruebas se ha desarrollado correctamente, se crearán todas las entradas posibles del programa, combinaciones aunque no conlleve realizarlas para una prueba adecuada. Otra manera es indicando la distribución estadística que siguen. La frecuencia de entrada sea la adecuada

para orientar pruebas hacia aquello que es más probable que se desarrolle en la práctica real. Esta forma de diseñar casos de prueba es menos utilizada que las técnicas de caja negra y caja blanca.

Como ejemplo se considera realizar una prueba de frontera para una aplicación que verifica si una secuencia de cuatro números reales X1, X2, X3, X4 es un dato válido. Suponemos que los datos válidos significa que:

$$-5 < X_1 \leq 10$$
$$X_1 + X_2 \leq X_3$$
$$X_3 \geq X_4$$

Figura 126.Restricciones de prueba frontera

Las pruebas de frontera requieren la selección de los datos de prueba con las siguientes restricciones.

X1 = -5(ilegal)
X1=10
X1+X2 =X3
X3=X4

Figura 127.Restricciones para datos de prueba

Como podemos observar X1 debe tener solo dos valores, X3 se determina una vez seleccionados X1 y X2, y X4 se determina por X3. Debe elegirse a partir de un número infinito de valores de X2 y estos valores se eligen al azar para evitar el sesgo.

4.4.8.6 Basadas en el código

4.4.8.6.1 Criterio basado en el Flujo de Control

- Pruebas de condición

  Las condiciones en una sentencia pueden ser: simples o compuestas. Una condición simple puede ser una variable lógica (verdadera o falsa) o una expresión relacional de la siguiente forma:

E1 (operador relacional) E2,
Donde E1 y E2 son expresiones aritméticas y el operador relacional es del tipo **<, ≤, >,≥ , =, ?#.**

Figura 128.Expresión relacional

Las condiciones compuestas están formadas por varias condiciones simples, operadores lógicos del tipo NOT, AND, OR y paréntesis
Los errores más comunes que se producen en una condición y que por lo tanto hay que probar son:

- Error en el operador lógico: que sean incorrectos, desaparecidos, sobrantes, etc.
- Error en la variable lógica.
- Error en la expresión aritmética.
- Error en el operador relacional.
- Error en los paréntesis.

En las decisiones, es necesario hacer pruebas de ramificaciones, que consisten en probar la rama verdadera y la rama falsa y cada condición simple.

En general, existen los siguientes tipos de pruebas relacionadas con las condiciones y decisiones:

- De cobertura de decisión.
- De cobertura de condición.
- De cobertura de decisión/condición.
- Se generan a condición los casos de prueba necesarios para obtener una cobertura completa de decisiones:

  D1 -> (h ≥ 0) and (h ≤ 23)
  D2 -> (m ≥ 0) and (m ≤ 59)
  D3 -> (s ≥ 0) and (s ≤ 59)

  donde,

  Di= decisiones
  h=hora
  m= minutos
  s= segundos

Cada decisión debe tomar al menos una vez el valor verdadero y otra el valor falso:

	Valor verdadero	Valor falso
D1	h=10	h=24
D2	m=30	m=60
D3	s=59	s=70

Tabla 189.Decisión que debe tomar al menos una vez el valor verdadero y el valor falso

Cubrir todas las decisiones, se definen los siguientes casos:

Caso prueba 1: D1 = verdadero; D2 = verdadero; D3 = verdadero
h=10; m=20; s=34

Caso prueba 2: D1 = verdadero; D2 = verdadero; D3 = falso
h=10; m=20; s=67

Caso prueba 3: D1 = verdadero; D2 = falso
h=10; m=60

Caso prueba 4: D1 = falso
h=24

Figura 129.Decisiones tomadas mediante casos

- Prueba del camino

Se define para este tipo de pruebas un conjunto básico de caminos de ejecución usando una medida calculada previamente de la complejidad del módulo llamada complejidad ciclomática propuesta por McCabe, se basa en el grafo de flujo.

La complejidad ciclomática indica el número de caminos básicos a probar y responde a la siguiente fórmula:

$$V(G) = Aristas - Nodos + 2$$

Los pasos a seguir para realizar las pruebas de camino básico son:

1. Dibujar el grafo de flujo.
2. Determinar la complejidad ciclomática del grafo.
3. Determinar los caminos linealmente independientes.
4. Preparar los casos de prueba que forzarán la ejecución de cada camino.

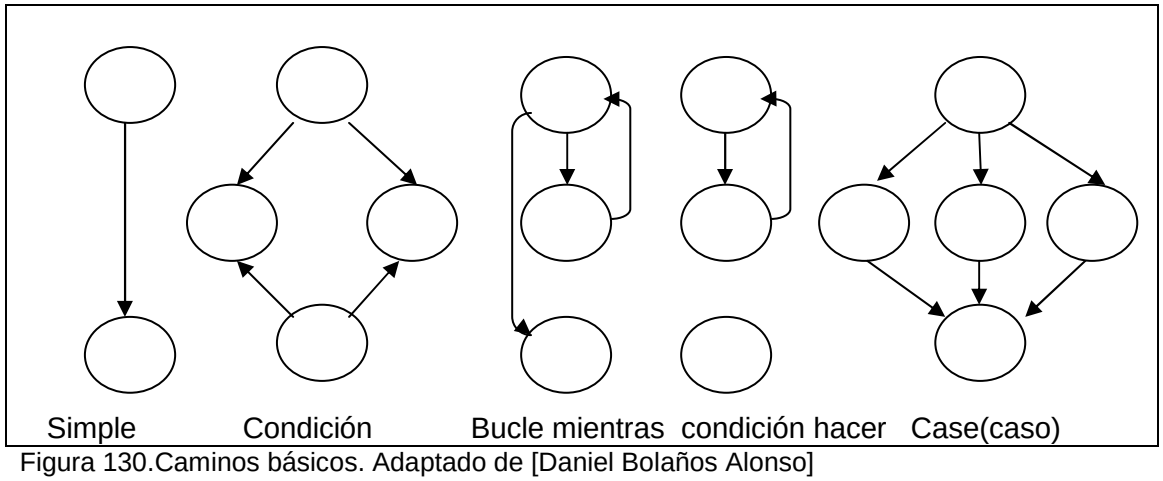


Figura 130.Caminos básicos. Adaptado de [Daniel Bolaños Alonso]

Existen correlaciones entre la complejidad de un módulo y el número de errores en el módulo, el tiempo para encontrar errores, el esfuerzo de prueba y el esfuerzo de mantenimiento. Un módulo con complejidad ciclomática mayor de 10 debe ser revisado para posibles simplificaciones o contemplar la posibilidad de subdividirlo en varias rutinas.

Utilización de la complejidad ciclomática de McCabe

La métrica de McCabe ha sido muy popular en el diseño de pruebas desde su presentación. Esta métrica es un indicador del número de caminos independientes que existen en un grafo, un camino independiente es: como una idea basada en los conceptos matemáticos de base de vectores y vectores linealmente independiente.

McCabe lo definió como un buen criterio de prueba la consecución de la ejecución de un conjunto de caminos independientes, lo que implica probar un número de caminos igual al de la métrica.

La complejidad de McCabe V (G) se puede calcular de las siguientes tres maneras a partir de un grafo de flujo G:

1. $V(G) = a - n + 2$, siendo “a” el número de arcos o arista del grafo y n el número de nodos.
2. $V(G) = r$, siendo r el número de regiones cerradas del grafo
3. $V(G) = c + 1$, siendo c el número de nodos de condición

Figura 131.Manera de calcular un grafo de flujo G a partir de la complejidad de McCabe

La experimentación con la métrica de McCabe ha dado como resultado las siguientes conclusiones:

- V(G) marca un límite mínimo de número de casos de prueba para un programa, contando cada condición de decisión como un nodo individual.

- V(G) es mayor que 10 la probabilidad de defecto en el módulo o en el programa crece bastante si dicho valor alto no se debe a sentencias case-if (caso-si) o similares. En estos casos es recomendable replantear el diseño modular obtenido.

4.4.8.6.2 Criterio basado en el Flujo de Datos

Las pruebas de flujo de datos es un término desafortunado, porque la mayoría de los desarrolladores de software piensan en algún tipo de conexión con los diagramas de flujo de datos. Las pruebas de flujo de datos se refieren a la

forma de las pruebas estructurales que se centran en los puntos en que las variables reciben valores y los puntos en que estos valores se utilizan (o referencian).

Se estudian dos formas principales de las pruebas de flujo de datos:

- Ofrece un conjunto de definiciones básicas y una estructura unificadora de las métricas de cobertura de la prueba.
- Se basa en un concepto llamado "corte del programa".

La mayoría de los programas ofrecen una funcionalidad en términos de datos. Las variables que representan los datos, reciben valores, y estos valores se utilizan para calcular los valores de otras variables.

En los programas se analiza el código fuente en cuanto a las declaraciones que las variables reciben; los valores y declaraciones en los que se utilizan estos valores. Muchas veces, los análisis se basan en las concordancias de los números de instrucciones en los que ocurren los nombres de variables.

Los principios del flujo de datos se analizan a menudo centrándose en una serie de errores que se conocen como anomalías de referencia entre las cuales se encuentran:

- Una variable que se define, pero nunca se utiliza (de referencia).
- Una variable que se define dos veces antes de que se utilice.

Cada una de estas anomalías puede ser reconocida a partir de la concordancia de un programa. Dado que la información es la concordancia generada por el compilador, estas anomalías pueden ser descubiertas por lo que se conoce como análisis estático: la búsqueda de fallos en el código fuente sin ejecutarla.

4.4.8.6.3 Modelos de referencia para pruebas basadas en código

El grafico de flujos muestra el flujo de control lógico. Cada constructo estructurado tiene un correspondiente símbolo de grafico de flujo es decir un componente es un bloque de construcción de cómputo.

Los constructos estructurados en un gráfico de programa forman lo siguiente:

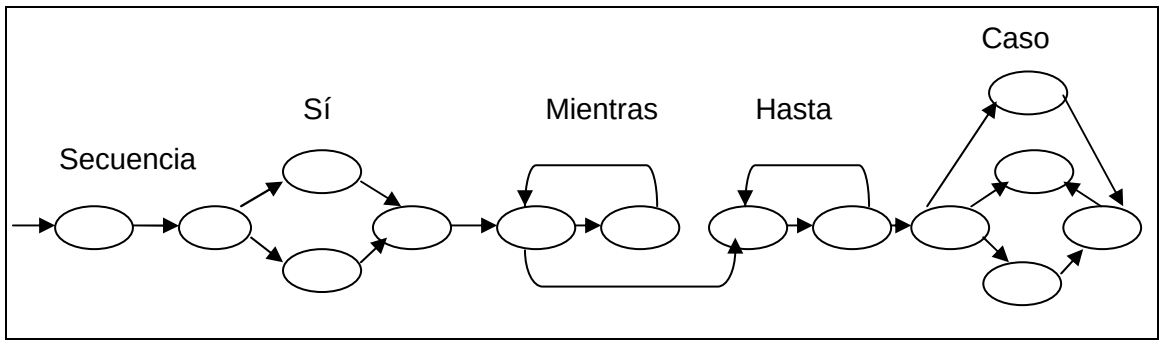


Figura 132.Gráfico de programa. Adaptado de [Roger S. Pressman]

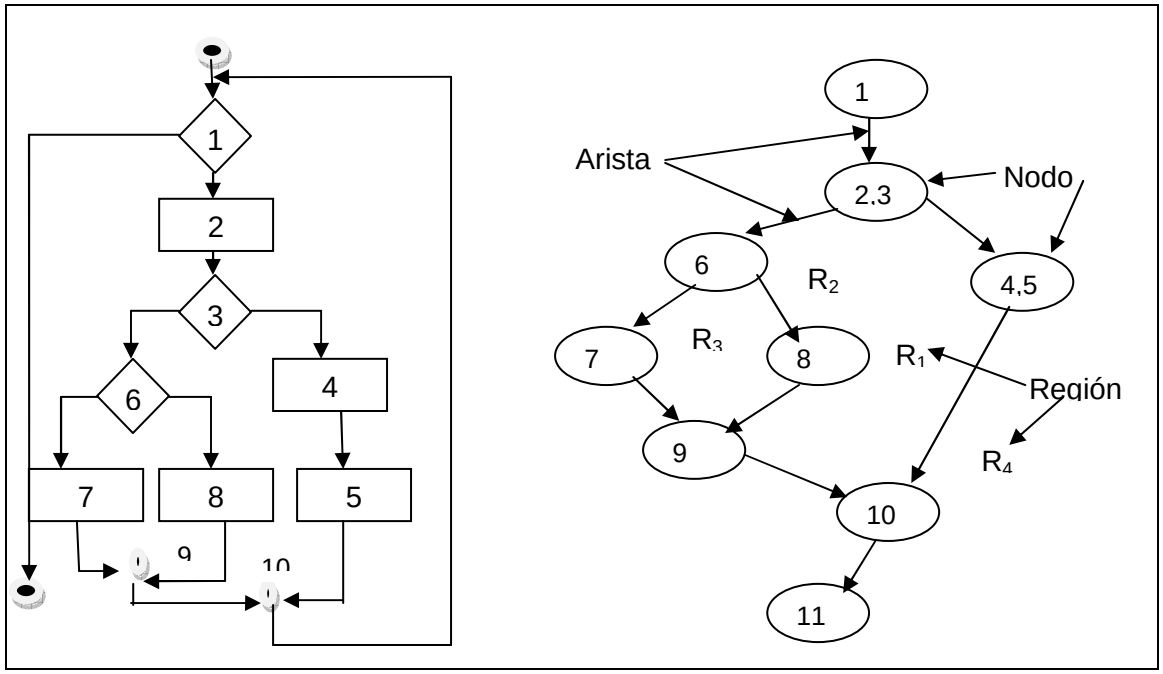


Figura 133.Diagrama de flujo y gráfico de flujo. Adaptado de [Roger S. Pressman]

La figura 133, del lado izquierdo muestra la estructura control del programa y la figura derecha mapea el diagrama de flujo en un gráfico de flujo. Cada círculo, llamado nodo de gráfico de flujo, representa uno o más enunciados de procedimiento.

Una secuencia de cajas de proceso y un diagrama de decisión pueden mapearse en un solo nodo. Las flechas en el gráfico de flujo se llaman aristas o enlaces, representan flujo de control y son semejantes a las flechas en el diagrama de flujo. Una arista debe terminar en un nodo, es decir si el nodo no representa algún enunciado de procedimiento.

Las áreas acotadas por aristas y nodos se llaman regiones. Cuando en un diseño de procedimiento se encuentran condiciones compuestas, la generación de un gráfico de flujo se vuelve más complicada. Una condición compuesta ocurre cuando uno o más operadores booleanos se representan en un enunciado condicional.

4.4.8.7 Basada en errores

4.4.8.7.1 Conjeturar errores

Consiste en enumerar una lista de errores posibles que pueden cometer los desarrolladores. Se generan casos de prueba en base a dicha lista. Esta técnica se ha denominado generación de casos o valores especiales ya que no se obtienen en base a otros métodos mediante la intuición o experiencia.

A continuación especificaremos algunos valores a tener en cuenta:

- Valor 0: Valor propenso a error tanto en la entrada como en la salida.
- Introducción de un número variable de valores como ejemplo podemos tomar una lista, es conveniente centrarse en el caso de no introducir ningún valor y el de un solo valor.
- Es importante suponer que el programador haya podido malinterpretar algo en la especificación.
- Es recomendable suponer lo que el usuario pueda introducir en la entrada de un programa. Para ello se debe prever toda clase de situaciones posibles.

4.4.8.8 Basadas en el uso

4.4.8.8.1 Perfil operativo

Recordemos que se produce un error cuando hay un fallo que se ejecuta. La idea de la prueba es ejecutar los casos de prueba de tal manera que, cuando se produce un fallo, la presencia de un fallo se pone de manifiesto. Podemos hacer una distinción importante: la distribución de los fallos en un sistema está indirectamente relacionada con la fiabilidad del sistema. La visión más simple de la fiabilidad del sistema es la probabilidad de que no se produce un fallo durante un intervalo de tiempo específico.

La idea de los perfiles de funcionamiento consiste en determinar las frecuencias de ejecución de varios hilos, y utilizar esta información para seleccionar temas para probar el sistema. Sobre todo cuando el tiempo de prueba es limitado, los perfiles operativos maximizan la probabilidad de encontrar defectos mediante la inducción de los fallos en los hilos con mayor frecuencia.

Una forma de determinar el perfil de funcionamiento de un sistema es utilizar un árbol de decisión. Esto funciona especialmente bien cuando el comportamiento del sistema se basa en las máquinas de estado jerárquico. Para cualquier estado, nos encontramos (o estimamos) la probabilidad de cada transición de salida (la suma de estos debe ser 1). Cuando un estado se descompone en un nivel inferior, las probabilidades en el nivel inferior se convierten en los límites de división en el nivel superior.

Los perfiles operativos proporcionan una idea de movimientos de combinación de un sistema de entrega. Esto es útil por razones que no sólo optimizan las pruebas del sistema. Estos perfiles también se pueden utilizar en conjunto con simuladores para obtener una indicación temprana de rendimiento del tiempo de ejecución y la capacidad de operación del sistema.

Muchas veces, los clientes son una buena fuente de información de movimientos de combinación, porque se hace un intento de reproducir la realidad de un sistema de entrega.

El perfil operacional del software refleja cómo se utilizará este en la práctica. Consiste en la especificación de clases de entradas y la probabilidad de su ocurrencia. Cuando un nuevo sistema software reemplaza a un sistema existente manual o automatizado.

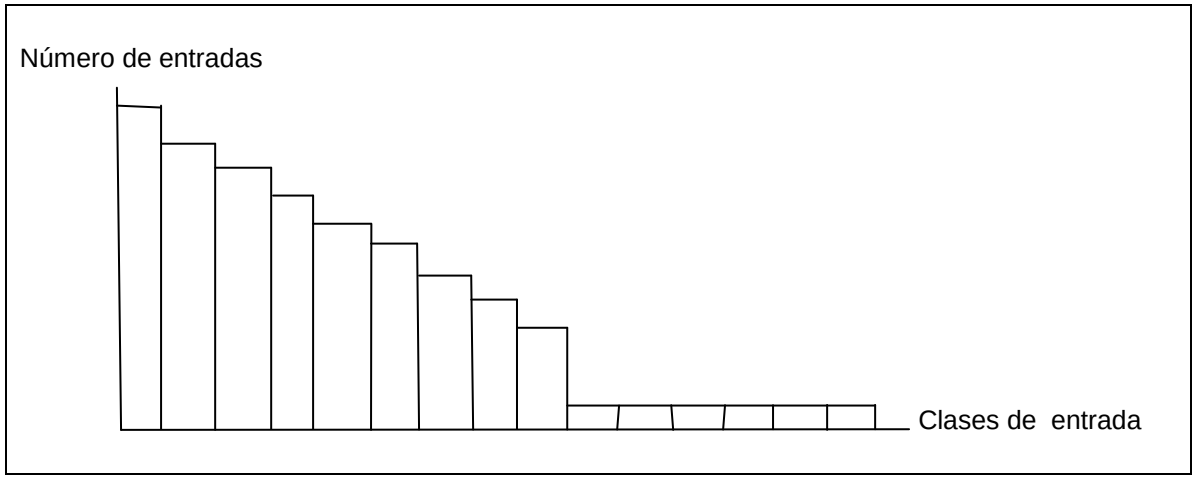


Figura 134.Perfil operacional. Adaptado de [Ian Sommerville]

En la figura 134 se concentran en un número pequeño de clases. Hay un número grande de clases en las que las entradas son altamente improbables, pero no imposibles.

El problema se complica debido a que los perfiles operacionales pueden cambiar a medida que se utilice el sistema. A medida que los usuarios comprendan el nuevo sistema y confíen más en él. Debido a estas dificultades *Hamlet sugiere que a veces es imposible desarrollar un perfil operacional fiable. Si el usuario no está seguro de que su perfil operacional es correcto, entonces no puede confiar en la exactitud de sus mediciones de fiabilidad*

4.4.8.8.2 Pruebas Orientadas a la Confiabilidad del Software

La meta de todo tipo de pruebas es la mejora de la fiabilidad del programa, pero si los objetivos del programa incluyen declaraciones específicas sobre la confiabilidad, las pruebas específicas de fiabilidad pueden ser desarrolladas. Las pruebas en los objetivos de confiabilidad pueden ser difíciles. Por ejemplo, en una red de área extensa (WAN), un proveedor de servicios de Internet tiene generalmente un tiempo de actividad específica del 99,97 por ciento durante la vida útil del sistema.

No se conoce ninguna manera de que se podría poner a prueba este objetivo con un período de prueba de meses o incluso años. Los sistemas críticos en el software de hoy en día tienen estándares de confiabilidad, y el hardware de hoy posiblemente se podría esperar para apoyar estos objetivos.

Por ejemplo, si el área de prueba del programa es de gran importancia, agregamos un nuevo concepto denominado afirmaciones de inducción. El objetivo de este método es el desarrollo de un conjunto de teoremas sobre el programa en cuestión, la prueba garantiza la ausencia de errores en el programa. El método comienza por escribir afirmaciones acerca de las condiciones de entrada del programa y resultados correctos. Las afirmaciones se expresan simbólicamente en un sistema de lógica formal, por lo general cálculo de predicados. A continuación, localizar cada bucle en el programa y, para cada bucle, escribir una afirmación que indica el invariante.

El programa ahora se ha dividido en un número fijo de caminos de longitud fija (todos los caminos posibles entre un par de afirmaciones). Para cada ruta, a continuación, tomar la semántica de los enunciados del programa, intervenir para modificar la afirmación, y, finalmente, llegar al final de la ruta. En este punto, existen dos afirmaciones al final del camino: la original y la que se deriva de la afirmación en el extremo opuesto.

A continuación, se escribe un teorema que indica que la aseveración original implica la afirmación de derivados, y el intento de demostrar el teorema. Si los teoremas se pueden demostrar, se podría asumir que el programa está libre de errores, siempre y cuando el programa termina con el tiempo. Una prueba independiente es necesaria para demostrar que el programa siempre termine con el tiempo previsto.

Las pruebas de fiabilidad y, de hecho, el concepto de ingeniería de la fiabilidad del software (SRE), están hoy cada vez más presentes y son cada vez más importantes para los sistemas que deben mantener tiempos de actividad muy altos.

Un modelo de crecimiento de fiabilidad es un modelo de cómo cambia la fiabilidad del sistema a lo largo del tiempo durante el proceso de pruebas. A medida que se descubren los fallos en el sistema, los defectos subyacentes que provocan estos fallos son reparados para que la fiabilidad del sistema mejore durante las pruebas y depuración.

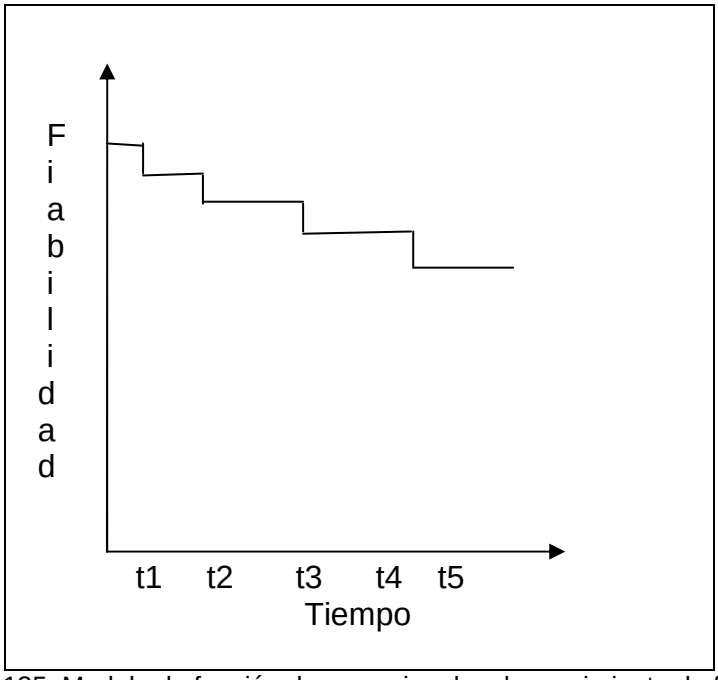


Figura 135. Modelo de función de pasos iguales de crecimiento de fiabilidad. Adaptado de [Ian Sommerville]

En la figura 135, los periodos de tiempo sobre el eje horizontal reflejan el tiempo entre entregas del sistema para pruebas, de forma que tienen longitudes diferentes.

4.4.8.9 Basadas en la Naturaleza de la Aplicación

4.4.8.9.1 Pruebas Orientadas a Objetos

Los niveles o etapas de las pruebas orientadas a objetos se pueden clasificar en los siguientes pasos:

- Pruebas de los métodos y operaciones individuales de las clases.
- Pruebas de las clases individuales.
- Pruebas de agrupación de objetos.
- Pruebas del sistema entero.

Ian Sommerville define un objeto como una entidad que tiene un estado y conjunto de operaciones definidas que operan sobre ese estado. El estado se representa como un conjunto de atributos del objeto.

Las operaciones que son asociadas al objeto proveen servicios a otros objetos, denominados clientes que solicitan dichos servicios cuando se requiere llevar a cabo algún cálculo. Los objetos se crean conforme a una definición de clases de objetos. Una definición de clases de objetos sirve como plantilla para crear objetos. Esta incluye las declaraciones de todos sus atributos y operaciones asociadas con un objeto de esa clase.

Las pruebas orientadas a objetos siguen las pautas expuestas anteriormente. Se prueban clases u objetos pertenecientes a clases que en muchas ocasiones están constituidos por más de una función.

1. Pruebas de clases de objetos

   El objetivo de estas pruebas es asegurar que una clase y todas sus instancias cumplen con el comportamiento perfilado en la definición de requisitos realizada en la fase de análisis del ciclo de vida de desarrollo. Este proceso consiste en ejecutar todas las instrucciones de un programa al menos una vez y ejecutar todos los caminos del programa.

   - Para cubrir las pruebas es necesario lo siguiente:
   - Probar todas las operaciones asociadas a los objetos de forma aislada.
   - Probar todos los atributos asociados al objeto.
   - Ejecutar los objetos en todos los estados posibles. Para ello se simulan todos los eventos que provocan un cambio de estado en el objeto en cuestión.

2. Pruebas de integración de objetos

   Consisten en asegurar que las clases, y sus instancias conforman un software que cumple con el comportamiento definido. Para ello, se realizan pruebas que verifican que todas las partes del software funcionan juntas de la forma definida en especificación de requisitos.

Tipo de prueba	Descripción
Basado en escenarios	Se definen los casos de uso o escenarios que especifican como se utiliza el sistema. A partir de aquí se empieza probando los escenarios más probables, siguiendo con los menos frecuentes y terminando con los escenarios excepcionales
Basado en subprocesos o cadena de eventos	Se basan en probar las respuestas obtenidas al introducir una entrada específica o un evento del sistema o un conjunto de eventos de entrada

Tabla 190.Tipos de prueba en la integración de objetos

3. Pruebas de interfaces

   Es conveniente hacer uso de la heurística maliciosa pasando por ejemplo, parámetros con apuntadores nulos, variar el orden de objetos que interactúan mediante memoria compartida. Además se deben diseñar casos de prueba en que los valores de los parámetros sean valores frontera (los primeros y últimos de su rango)

Desde el punto de vista de diseño de casos de prueba:

- Técnicas de caja negra

  Tiene validez que su uso se centra en el comportamiento de la aplicación. En el caso de pruebas de sistema, son las descripciones de casos de uso las que aportan referencia sobre la manera de diseñar los casos de uso. Los escenarios de casos de uso permiten definir los casos de prueba aplicando técnicas de caja negra sobre los datos y eventos incluidos en los escenarios y a su vez identifican en los diagramas de actividad, secuencia, colaboración, diagramas de estado y todos los flujos de error y excepción.

- Técnicas de caja blanca

  Disminuyen sus posibilidades de aplicación, ya que quedan recluidas en las instrucciones de cada método de cada clase para conseguir el nivel de cobertura más adecuado. La cobertura de sentencias resulta primordial, el tratamiento de bucles, decisiones a través de las coberturas de condiciones y decisiones es más que recomendable.

Hay dos cuestiones que cabe mencionar en las pruebas orientadas a objetos:

- Uno de los problemas que provoca la tecnología de objetos en las pruebas es el tratamiento de la herencia.
- La propia programación o diseño orientados a objetos contemplan conceptos de: encapsulación, herencia, polimorfismo etc

4.4.8.9.2 Basadas en Componentes

Denominada prueba de función, permite un enfoque mediante un conjunto de pruebas que intentan descubrir errores en funciones webapps (web del conjunto de aplicaciones de productividad. Las aplicaciones web permiten a los usuarios acceder a sus documentos directamente desde cualquier parte dentro de un navegador web así como compartir archivos y colaborar con otros usuarios en línea).

Cada función webapps es un componente del software implantado en uno o varios lenguajes de programación y se prueban utilizando técnicas de caja negra y en algunos casos de caja blanca. A continuación especificaremos los casos de prueba en los siguientes métodos aunque los hemos explicado con más detalle en apartados anteriores:

Método	Descripción
Partición de equivalencia	Los casos de prueba para cada clase de entrada en el dominio se derivan y ejecutan mientras que otras clases de entrada se mantienen constantes. Ejemplo: aplicación de comercio electrónico. Los casos de prueba se diseñan con la intención de descubrir errores.
Análisis de valor de límite	Los datos de los formularios se prueban en sus límites.
Prueba de rutas	Si la complejidad lógica (puede determinarse al calcular la complejidad ciclomática del algoritmo) es alta pueden utilizarse las pruebas de rutas(método de diseño caja blanca) para garantizar cada ruta independiente en el programa.

Tabla 191.Algunos métodos usados en pruebas a nivel de componentes

Cada caso de prueba en el nivel de componentes especifica todos los valores de entrada y salida que se espera que proporcione el componente. La salida real producida como consecuencia de la prueba se registra para futuras referencias durante el soporte y mantenimiento.

4.4.8.9.3 Pruebas para Internet

Las aplicaciones de Internet son esencialmente aplicaciones cliente-servidor en las que el cliente es un navegador web y el servidor es un servidor web o aplicación.

La complejidad de estas aplicaciones es muy dispar. Algunas empresas han construido aplicaciones para el negocio-a-consumidor y utilizan como servicios bancarios o tiendas al por menor, mientras que otros tienen las aplicaciones de negocio como la gestión de la cadena de suministro. Desarrollo y presentación de usuario/estrategias de interfaz de usuario varían para los distintos tipos de sitios web, y por lo tanto el enfoque de las pruebas varía para los diferentes tipos de sitios.

El objetivo de las pruebas de aplicaciones basadas en Internet no es diferente de la de las aplicaciones tradicionales. Es necesario para descubrir errores en la aplicación antes de implementarla en Internet. Dada la complejidad de estas aplicaciones y la interdependencia de los componentes, es probable que tenga éxito en la búsqueda de muchos errores.

La importancia de erradicar los errores en una aplicación de Internet no puede ser subestimada. Como resultado de la apertura y accesibilidad de Internet, la competencia en el área de los negocios-a-consumidor es intensa. Por lo tanto, la Internet ha creado un mercado de compradores de bienes y servicios. Los consumidores han desarrollado grandes expectativas, y si su sitio no se carga rápidamente, responde de inmediato, y proporciona funciones de navegación intuitiva, lo más probable es que el usuario va a encontrar otro sitio con el que hacer negocios.

Parece que los consumidores tienen mayores expectativas de calidad para las aplicaciones de Internet. Una de las razones de este comportamiento es que han pagado una aplicación y debe ser un producto que ellos perciben como útil o conveniente. Incluso un programa menos que satisfactorio, no se puede corregir fácilmente, así que si la aplicación por lo menos satisface las necesidades básicas de los usuarios, es probable mantener el programa. En Internet, una aplicación de calidad media es probablemente la causa del cliente para el uso del sitio de un competidor.

No sólo los clientes salen de su sitio web si presenta mala calidad, su imagen corporativa se empaña así. En general, los consumidores no pagan para acceder a su sitio web, así que hay poco incentivo a permanecer fieles en la cara del diseño web mediocre o de rendimiento.

Arquitectura básica de comercio electrónico

Se ofrecerá una visión general de los tres niveles de cliente-servidor (C / S) de la arquitectura usada en Internet básica de comercio electrónico. Cada nivel es tratado como un cuadro negro con interfaces bien definidas. Este modelo permite cambiar el funcionamiento interno de cada nivel sin preocuparse de romper otro nivel.

Los navegadores en la forma de representar el contenido de un sitio web. Las pruebas de compatibilidad de navegadores es uno de los retos asociados con las aplicaciones de Internet de prueba. Los clientes utilizan las aplicaciones personalizadas y utilizan Internet como canal a un sitio determinado. En este escenario, la aplicación simula un estándar de aplicación cliente-servidor que se puede encontrar en la red de una empresa de área local.

Nivel	Descripción
Nivel 1 : Servidor web	Representa el primer nivel de la arquitectura de tres niveles y alberga el sitio web. La apariencia de una aplicación de Internet proviene de los primeros niveles. Por lo tanto, otro término para este nivel es la capa de presentación porque proporciona el contenido visual para el usuario final. El servidor Web puede utilizar estática HyperText Markup Language (HTML) o Common Gateway Interface (CGI) para crear scripts HTML dinámico, pero lo más probable es que utiliza una combinación de páginas estáticas y dinámicas.
Nivel 2: capa de negocios	Alberga la aplicación del servidor. Se ejecuta el software de los modelos de sus procesos de negocio. A continuación se enumeran algunas de las funcionalidades asociadas con la capa de negocio: - Procesamiento de transacciones. - Autenticación de usuario. - Validación de datos. - Solicitud de registro.
	Se centra en el almacenamiento y recuperación de datos desde una fuente de datos, por lo general un sistema de base de datos relacional (RDBMS). Este nivel se compone de una infraestructura de base de datos para comunicarse con el segundo nivel. La interfaz en la capa de datos está definida por el modelo de datos, que

Nivel 3: Capa de datos	describe cómo desea almacenar los datos. Por lo general, ajusta los sistemas de base de datos en esta capa para manejar las tasas de transacción elevadas que se encuentran en un sitio de comercio electrónico. Algunos sitios de comercio electrónico pueden colocar un servidor de autenticación en esta capa. Muy a menudo, se utiliza un servidor LDAP (Protocolo Ligero de directorio de aplicaciones) del servidor para esta función.

Tabla 192.Tres niveles de cliente-servidor (C/S)

Problemas de pruebas

Los problemas de pruebas se enfrentan a muchos retos en el diseño y prueba de aplicaciones basadas en Internet, debido a la gran cantidad de elementos que no pueden controlarse y el número de componentes interdependientes. Una aplicación basada en Internet tiene muchos puntos de fallos que se deben tener en cuenta al diseñar un método de prueba.

A continuación se especifican algunos ejemplos de problemas de pruebas de aplicaciones basadas en internet:

Nombre	Descripción
Amplia variedad de usuarios diversos	Los usuarios del sitio web poseen diferentes conjuntos de capacidades, emplean una variedad de navegadores, y el uso de diferentes sistemas operativos o dispositivos. También los clientes pueden acceder al sitio web utilizando una amplia variedad de velocidades de conexión. No todo el mundo tiene acceso a Internet o banda ancha.
El entorno de negocios	Si se opera en un sitio de comercio electrónico, entonces se deben considerar cuestiones tales como: Cálculo de los impuestos, la determinación de los costos de envío, completar las transacciones financieras, y el seguimiento de perfiles de clientes.
Locales	Los usuarios pueden residir en otros países, en cuyo caso tendrán problemas de internacionalización como la traducción de la lengua, las consideraciones de la zona horaria, y la conversión de moneda.
Prueba de entornos	Para probar correctamente su solicitud, tendrá que duplicarse el entorno de producción. Esto significa que deben utilizarse los servidores Web, servidores de aplicaciones y servidores de base de datos que son idénticos a los equipos de producción. Para que los resultados de pruebas sean más precisos, la infraestructura de la red tendrá que ser duplicada también. Esto incluye routers, switches y cortafuegos.
	Debido a que el sitio está abierto al mundo,

Seguridad	debe protegerse de los piratas informáticos. Se puede traer a el sitio web un paro de denegación de servicio (DoS) o estafar y obtener información de sus clientes a través de tarjetas de crédito.

Tabla 193.Lista que contiene algunos ejemplos de los problemas asociados con las pruebas de aplicaciones basadas en Internet

Otro reto importante de pruebas que se enfrenta hoy en día son pruebas de compatibilidad del navegador. Hay varios navegadores diferentes en el mercado, y cada uno se comporta de forma diferente. Aunque existen normas para el funcionamiento del navegador, la mayoría de los proveedores mejoran sus navegadores para tratar de atraer a una base de usuarios.

Aunque existen muchos problemas al probar aplicaciones basadas en Internet, conviene reducir esfuerzos de pruebas para áreas específicas. Algunas de las pruebas se centran en cuestiones de usabilidad y factores humanos. Esta área se centra en la apariencia de su aplicación. Elementos como fuentes, colores y gráficos juegan un papel importante en que los usuarios aceptan o rechazan su solicitud.

El rendimiento del sistema también influye en la primera impresión de un cliente. Como se mencionó anteriormente, los usuarios de Internet desean una gratificación instantánea. No van a esperar mucho tiempo para cargar las páginas o las transacciones para completar. Literalmente, el retraso de unos segundos puede inducir al cliente a probar otro sitio. El bajo rendimiento también puede inducir a los clientes a dudar de la fiabilidad de un sitio. Por lo tanto, se deben establecer objetivos de rendimiento y pruebas de diseño que revelan los problemas que causa el sitio.

Los usuarios exigen que la transacción se produzca con rapidez y precisión en la compra de productos o servicios del sitio. Su aplicación es probable que recoja datos para completar tareas tales como compras o registros de correo electrónico. Por lo tanto, debe asegurarse de que los datos que recopila son válidos.

Capa de presentación	Capa de Negocio	Nivel de datos
• Asegúrese de que las fuentes son las mismas en todos los navegadores • Asegúrese de que todos los enlaces apuntan a archivos válidos o sitios web. • Revise los gráficos para asegurarse de que son la resolución y el tamaño correctos. • Revisar la ortografía de cada página. • Permitir que un editor de textos pueda revisar la gramática y el estilo.	• Comprobar si el cálculo de impuesto sobre las ventas y los gastos de envío son los adecuados. • Asegúrese de documentar las tasas de rendimiento que cumplan los tiempos de respuesta y las tasas de rendimiento. • Verificar que las transacciones se completan correctamente. • Asegurar que los datos se recogen correctamente.	• Garantizar las operaciones de base de datos de cumplir con las metas de desempeño • Verificar que los datos se almacenan correctamente y con precisión • Comprobar que se pueden recuperar copias de seguridad actuales. • Prueba de fallos o las operaciones de redundancia.

Tabla 194.Ejemplos de la presentación, de negocios, y prueba de nivel de datos

En el entorno de Internet, es fundamental mantener el sitio web disponible para uso del cliente. Esto requiere desarrollar e implementar guías de mantenimiento para todas las aplicaciones y servidores de apoyo. Deben supervisarse los registros, los recursos del sistema, y copias de seguridad para evitar que dichos elementos críticos no fallan. Se debe maximizar el tiempo medio entre fallos (MTBF) y minimizar el tiempo medio de recuperación (MTTR) para estos sistemas.

La conectividad de red proporciona otra área en la que centrar los esfuerzos de prueba. En algún momento, nos podemos encontrar con conectividad de red a la baja. El origen del fallo puede ser la propia Internet, el proveedor de servicios, o la red interna. Por lo tanto, es necesario crear planes de contingencia para su aplicación y la infraestructura para responder correctamente cuando se produce un corte.

Las estrategias de evaluación

Desarrollar una estrategia de pruebas para aplicaciones basadas en Internet requiere de una sólida comprensión de cada uno de los componentes de hardware y software que componen la aplicación. Como es crítico para las pruebas con éxito de las aplicaciones estándar, se requiere de un documento de especificaciones que se necesita para describir la funcionalidad esperada y

el rendimiento de su sitio web. Sin este documento, no se pueden diseñar las pruebas adecuadas.

Es necesario poner a prueba los componentes desarrollados internamente y los adquiridos por un tercero. Para los componentes de desarrollo se deben incluir: la creación de la unidad, módulo de pruebas y la realización de revisiones de código.

Deben integrar los componentes en su sistema sólo después de verificar que cumplen con las especificaciones de diseño y funcionalidad y se describen en el documento de especificación. La prueba de aplicaciones basadas en Internet es mejor que la de un enfoque de divide y vencerás. La arquitectura de las aplicaciones de Internet permite identificar áreas discretas de controles dirigidos.

Capas	Descripción
Capa de presentación	La capa de una aplicación de Internet que proporciona la GUI (interfaz gráfica de usuario).
Capa lógica de negocios	La capa de los modelos de sus procesos de negocio, tales como la autenticación de usuarios y transacciones.
Capa de acceso de datos	La capa de los datos de inicio utilizados por la aplicación o en la que se recogen datos de los usuarios finales.

Tabla 195.Tres niveles de aplicaciones cliente-servidor

En la tabla 195, cada nivel tiene sus propias características que fomentan la segmentación de la prueba. La prueba de cada nivel es de forma independiente en el cual permite identificar más fácilmente los errores antes de la prueba que se completa del sistema que se inicia.

A continuación explicaremos con más detalle como probar cada capa:

1. Capa a nivel de presentación

   La capa de presentación consiste en encontrar errores en la guía de interfaz de usuario (GUI). Esta capa es importante porque proporciona el encanto de la aplicación, el detectar y corregir errores en esta capa son críticos al presentar un enfoque de calidad. Las pruebas de capa a nivel de presentación son muy laboriosas. Se identifican tres grandes áreas de pruebas en capa de presentación:

   - Contenido de prueba: Estética general, las fuentes, el color, la ortografía, la exactitud del contenido, los valores por defecto.
   - Sitio web de arquitectura: Los enlaces rotos o gráficos.
   - Entorno de usuario: Versiones del navegador web y la configuración del sistema operativo.

Área	Comentarios
Usabilidad / factores humanos	Revisión general de apariencia. Las fuentes, colores y gráficos juegan un papel importante en la estética de la aplicación.
Rendimiento	Compruebe las páginas de carga rápida. Compruebe si hay transacciones rápidas. El rendimiento lento a menudo crea una mala impresión.
Las reglas de negocio	Verifique la representación exacta de los procesos de negocio. Considere el entorno empresarial para grupos de usuarios para establecer objetivos.
Transacción precisión	Asegúrese de completar transacciones con precisión Asegúrese de que las transacciones canceladas se deshacen correctamente.
Datos de validez y la integridad	Verifique los formatos válidos: número de teléfono, direcciones de correo electrónico, las cantidades de divisas, etc. Asegúrese del conjunto de caracteres adecuado.
La fiabilidad del sistema	Pruebe la capacidad de conmutación por error de la web, aplicaciones y servidores de bases de datos. Maximizar el tiempo medio entre fallos y minimizar el tiempo medio de reparación.
Arquitectura de red	La prueba de conectividad redundante. Prueba de comportamiento de las aplicaciones durante caídas de red.

Tabla 196.Elementos de prueba en cada nivel de la capa de presentación

2. Capa a nivel de negocio

La Prueba de la capa de negocio se centra en la búsqueda de errores, en la lógica de negocio de la aplicación de Internet. Es muy similar a las pruebas de aplicaciones independientes en las que se pueden emplear tanto técnicas de caja blanca como de caja negra. Se crean planes de pruebas y procedimientos que detectan errores en los requisitos de rendimiento de la aplicación, adquisición de datos y procesamiento de transacciones.

En los métodos de caja blanca se emplean componentes de desarrollo propio, ya que tienen acceso a la lógica del programa. Los métodos de

caja negra son técnicas de prueba con enfoque de las pruebas principales para esta capa. Independientemente de su enfoque, existen ciertas características de la aplicación que siempre deben probar.

Estas áreas son las siguientes:

Área	Comentario
Rendimiento	Prueba para ver si la solicitud cumple con las especificaciones de desempeño mediante la descripción documentada (por lo general se especifica en los tiempos de respuesta y las tasas de rendimiento).
Validez de los datos	Prueba para detectar errores en los datos obtenidos de los clientes.
Transacciones	Prueba para descubrir errores en el procesamiento de transacciones, que pueden incluir elementos tales como el procesamiento de tarjetas de crédito, verificación de correo electrónico, y el cálculo del impuesto, ventas, etc.

Tabla 197.Elementos de prueba en cada nivel de la capa de presentación

Pruebas de rendimiento

Una aplicación de Internet de bajo rendimiento genera dudas sobre su solidez en la mente de los usuarios. Cargas de páginas largas y transacciones lentas, son ejemplos típicos. Para ayudar a alcanzar niveles adecuados de rendimiento, es necesario asegurarse de que las especificaciones de funcionamiento estén escritas en la fase de recopilación de requisitos. Un enfoque común al realizar las pruebas de rendimiento es la prueba de esfuerzo. A menudo, el rendimiento del sistema se degrada hasta el punto de ser inutilizable cuando el sistema se sobrecarga de peticiones.

Esto puede hacer que el tiempo de los componentes sea sensible a las transacciones y esto conlleva al fracaso. Los conceptos sobre las pruebas de esfuerzo se aplican a las pruebas de rendimiento de capa de negocio. La prueba de esfuerzo consiste en la aplicación con varios accesos y transacciones simulando el punto de fallo para poder determinar si la aplicación cumple con sus objetivos de rendimiento. Las pruebas de esfuerzo de la aplicación también permiten investigar la robustez y la escalabilidad de su infraestructura de red.

Validación de datos

Una función importante de la capa de negocio consiste en asegurar que los datos que recogemos de los usuarios son válidos. Se deberán comprobar errores en la recolección de datos.

Pruebas de transacciones

Se deben procesar las transacciones correctamente el 100 por ciento del tiempo. No hay excepciones. Los clientes no van a tolerar transacciones fallidas.

Además de una mala reputación y pérdida de clientes, también puede incurrir en responsabilidades legales asociadas a las transacciones fallidas. Se puede considerar como prueba del sistema transaccional de la capa de negocio. En otras palabras, se prueba la capa de negocio de principio a fin, tratando de descubrir los errores. Se debe mantener un documento escrito que especifique exactamente lo que constituye una transacción. Para una aplicación típica de Internet, un componente de transacción es más que completar una transacción financiera (como el procesamiento de tarjetas de crédito).

Eventos típicos que un cliente realiza en una transacción incluyen los siguientes aspectos:

- Búsqueda de inventario.
- Recopilación de artículos que el usuario quiere comprar.
- La compra de artículos, que pueden incluir el cálculo de impuesto sobre las ventas y los gastos de envío, así como el procesamiento de las transacciones financieras.
- Notificar al usuario de la transacción que se completa, por lo general a través de e-mail.

Estos aspectos mencionados anteriormente son procesos de verificación de las operaciones internas, deben probarse los servicios externos, tales como la validación de tarjetas de crédito, la banca y verificación de la dirección.

Capa a nivel de datos

La prueba de la capa de datos consiste principalmente en probar el sistema de gestión de base de datos que utiliza la aplicación para almacenar y recuperar información.

Sitios más pequeños pueden almacenar datos en archivos de texto, pero más grandes, es decir los sitios más complejos utilizan todas las funciones de nivel empresarial de bases de datos. Dependiendo de sus necesidades, se pueden utilizar ambos métodos.

Uno de los mayores desafíos relacionados con las pruebas de esta capa es duplicar el entorno de producción. Deben utilizarse las plataformas de hardware y las versiones equivalentes de software para realizar pruebas válidas. Una vez que se obtienen los recursos, tanto financieros como de trabajo, se debe desarrollar una metodología para el mantenimiento de producción y pruebas sincronizadas. Al igual que con los demás niveles, deben buscarse errores en ciertas áreas cuando se prueba la capa de datos.

Estos incluyen los siguientes:

Área	Comentario
Tiempo de respuesta	La cuantificación de los tiempos de finalización de lenguaje de manipulación de datos (DML) (Structured Query Language) [SQL] inserciones, actualizaciones y eliminaciones), consultas (SELECT), y las transacciones.
Integridad de los datos	Verificar que los datos se almacenan correctamente y de forma precisa.
Tolerancia a fallos y recuperación	Maximizar el tiempo medio entre fallos (MTBF) y minimizar el tiempo medio de reparación (MTTR).

Tabla 198.Elementos de prueba en cada nivel de la capa de datos

Tiempo de respuesta

Se crea descontento por parte de los clientes, para ello es de sumo interés asegurarse que el sitio web responde de manera oportuna a las peticiones del usuario y los eventos. El tiempo de respuesta de las pruebas en esta capa no incluye las cargas de página, sino que se centra en la identificación de las operaciones de base de datos que no cumplan con los objetivos de rendimiento.

Al probar el tiempo de respuesta de nivel de datos, se pretende asegurar que las operaciones de base de datos individuales se produzcan con rapidez para que no haya operaciones de cuello de botella. Una operación de base de datos consiste en insertar, eliminar, actualizar o consultar. La medición del tiempo de respuesta, simplemente consiste en determinar la duración de cada operación.

El tiempo de respuesta de la capa de prueba a nivel de datos está plagado de retos. Para ello se debe tener un entorno de prueba que coincida con lo que se utiliza en la producción, de lo contrario se pueden obtener resultados de prueba no válidos. Además, se debe tener una comprensión completa del sistema de base de datos para asegurarse de que está correctamente configurado y funcionando de manera eficiente.

Integridad de los datos

Integridad de los datos de prueba es el proceso de encontrar datos inexactos en las tablas de la base de datos. Estas pruebas difieren de la validación de datos, de la conducta durante las pruebas de la capa de negocio. Las pruebas de validación de datos tratan de encontrar en la recopilación de datos. En las pruebas de integridad de los datos se trata de encontrar errores en la forma de almacenar datos.

Tolerancia a fallos y recuperación

Uno de los objetivos de las operaciones de base de datos, en general, es el de maximizar el tiempo medio entre fallos (MTBF) y minimizar tiempo medio de reparación (MTTR). Se deben encontrar los valores especificados en la documentación de los requisitos del sistema para el sitio de comercio electrónico. El objetivo cuando se prueba la robustez del sistema de base de datos es tratar de superar estos números.

Maximizar MTBF depende del nivel de tolerancia a fallos del sistema de base de datos. Que pueda tener una arquitectura de migración tras el error que permite transacciones activas para cambiar a una nueva base de datos cuando el sistema principal falla. En este caso, los clientes pueden experimentar una interrupción del servicio pequeña, pero el sistema debe seguir siendo útil. Otra posibilidad es que se construya la tolerancia a fallos en la aplicación de manera que una base de datos se caiga y afecte el sistema.

4.4.8.9.4 Pruebas para GUI

La característica principal de cualquier interfaz gráfica de usuario (GUI) es que está orientado a eventos. Los usuarios pueden causar cualquiera de varios eventos en cualquier orden. Aunque es posible crear aplicaciones GUI en secuencia de eventos, muchas interfaces gráficas de usuario son deficientes en este sentido.

Las aplicaciones de interfaz gráfica de usuario ofrecen un pequeño beneficio a los probadores: no hay necesidad de pruebas de integración. Las pruebas unitarias se suelen encontrar en el llamado nivel de botón, es decir, los botones tienen funciones, y éstas pueden ser probadas. La esencia de la prueba de nivel de sistema para las aplicaciones de interfaz gráfica de usuario es el ejercicio de la naturaleza orientada a eventos de la aplicación.

Desafortunadamente, la mayoría de los modelos de lenguaje de modelado universal (UML) son de poca ayuda en caso de sistemas accionados. La principal excepción es los modelos de comportamiento, específicamente los gráficos de estado (state charts) y su caso más simple, las máquinas de estados finitos.

Las pruebas de interfaz de usuario hombre–máquina. Existen dos roles bien diferenciados:

- Probar la interfaz de usuario para garantizar que funciona correctamente desde el punto de vista técnico y que cumple los estándares y requisitos definidos.
- Usabilidad de la interfaz. La usabilidad se deriva de una comunicación efectiva de la información. Por ende, es necesario tener en cuenta los requisitos de usuario y desarrollar la interfaz de usuario de acuerdo con ellos.

4.4.8.9.5 Pruebas para Sistemas de Tiempo Real

Los métodos de diseño de casos de prueba para sistemas en tiempo real a medida que pasa el tiempo siguen evolucionando, mediante el cual se presentan los siguientes enfoques:

Estrategia	Descripción
Prueba de tareas	Su principal propósito en las pruebas del software de sistemas para tiempo real es comprobar cada una de las tareas de forma independiente es decir cada tarea en las pruebas convencionales se diseñan y se ejecutan durante dichas pruebas. La prueba de tarea desarrolla errores en la función y en la lógica más no en temporización y comportamiento.
Pruebas de comportamiento	Es posible simular el comportamiento de un sistema de tiempo real y probar su comportamiento como consecuencia de eventos externos. Las tareas de análisis sirven para el diseño de los casos de prueba que se desean desarrollar cuando se construye el software en tiempo real. Podemos usar la técnica basada en particiones de clases o equivalencia estudiada en el apartado 4.4.8.2.1 es decir los eventos se categorizan para las pruebas y las interrupciones del sistema, cada una se prueba de manera independiente y el comportamiento del sistema ejecutable se prueba para detectar errores que ocurren como consecuencia del procesamiento asociado con dicho evento. El comportamiento del software se prueba para detectar errores de comportamiento.
Prueba intertarea	Separados los errores en las tareas individuales y comportamiento del sistema, las pruebas se cambian a los errores relacionados con el tiempo. Las tareas asíncronas se comunican mutuamente, se prueban con distintas tasas de datos y carga de procesamiento para determinar si ocurren errores de sincronización de intertarea
Prueba de sistema	Al unir el software con el hardware se amplía el rango de pruebas del sistema con la intención de manifestar errores en la interfaz software y hardware. Los sistemas en tiempo real procesan interrupciones. Por ende los probadores realizan una lista de las distintas interrupciones y procesamiento que ocurren como consecuencia de las interrupciones.

Tabla 199.Estrategias de prueba para sistemas en tiempo real

4.4.8.9.6 Pruebas para Sistemas de Seguridad Crítica

Se describe la organización, calendario, recursos, responsabilidades, herramientas, técnicas, y las metodologías utilizadas en el desarrollo de software de seguridad crítica.

Plan de organización y responsabilidades

Se describen las actividades de seguridad de software dentro de la organización en general y se describen las relaciones organizativas y funcionales relativas a cuestiones de seguridad de software, líneas de comunicación, y la autoridad.

La relación de las tareas de programa de software de seguridad para tareas de sistema de programa de seguridad. La relación entre las organizaciones que tengan la responsabilidad de las tareas de seguridad que afectan el software y la organización de la gestión del programa de seguridad de software. La autoridad de la gestión de programas de software de seguridad para imponer el cumplimiento de los requisitos de seguridad y las prácticas se describen.

Gestión del programa de seguridad de software incluye la responsabilidad de hacer lo siguiente:

- Elaborar el Plan.
- Obtener y asignar recursos para garantizar la aplicación efectiva del Plan.
- Coordinar las tareas de planificación de seguridad con otros componentes de la organización y funciones, tales como la seguridad de desarrollo, sistema de control de calidad de software, la fiabilidad del software, gestión de configuración software, V & V, y las pruebas de software.
- Coordinar las tareas de software de seguridad en el contexto general del programa de seguridad del sistema.
- Coordinar las cuestiones técnicas relacionadas con la seguridad del software con otros componentes del desarrollo.
  y apoyar la organización, con el patrocinador del proyecto, o con el cliente
- Asegurar que los registros se mantienen adecuados para documentar la realización de actividades de seguridad de software.
- Participar en las auditorías de la ejecución del plan de seguridad del software.
- Garantizar la formación de la seguridad y otro personal apropiado en los métodos, herramientas y técnicas utilizadas en tareas de seguridad de software.

Figura 136.Plan de gestión de programa de seguridad

Recursos

Se especifican los requisitos de recursos y la asignación de los recursos a las tareas de seguridad. El plan de gestión describe cómo el uso de recursos se efectuará durante la ejecución de programas de software de seguridad. En los recursos se incluyen, pero no se limitan a, presupuesto, horario, personal de seguridad, otros miembros del personal, equipo y apoyo a otros equipos y herramientas.

Cualificación y formación del personal

Se especificarán las condiciones requeridas para el personal que llevará a cabo, como mínimo, las siguientes tareas:

- Diseñar e implementar la seguridad-críticas partes del sistema.
- Realizar tareas de análisis de software de seguridad.
- Prueba de características críticas para la seguridad.
- Software de auditoría de seguridad, plan de implementación.
- Realizar el proceso de certificación.

Figura 137.Requisitos de formación continua y las modalidades de cumplimiento de los objetivos de formación del personal con el software de seguridad relacionados con las responsabilidades.

Ciclo de vida del software

Se identificarán el ciclo de vida del software que se utilizará y especificará la relación entre las tareas específicas de software de seguridad y las actividades incorporadas en el ciclo de vida seleccionando el software de la organización.

Requisitos de documentación

Se especifican los documentos para su preparación y contenidos. En el plan de gestión se especifica el cambio y proceso de aprobación de programas de seguridad relacionados con las partes de todos los documentos del proyecto, incluido el propio plan de gestión.

Como mínimo, los siguientes requisitos de documentación deberán cumplirse para todo el software de seguridad crítica:

Nombre	Descripción
Software de gestión de proyectos	Documentación de cómo el programa de seguridad del software se llevará a cabo, integrado y gestionado con otras actividades de desarrollo.
Software de gestión de configuración	Información sobre la gestión de la configuración del software relacionado con la seguridad y módulos de los documentos.
Software de control de calidad.	La información sobre el aseguramiento de la calidad del software de seguridad relacionados con los módulos y los documentos.
Los requisitos de software de seguridad	Especificación de los requisitos de seguridad que debe cumplir el software para evitar riesgos del sistema de control.
Diseño de software de seguridad	Descripciones de los elementos del diseño de software que satisfacen los requisitos de seguridad de software.

Metodología de desarrollo de software, normas, prácticas, métricas, y convenciones	Prácticas aprobadas y controladas, que son esenciales para satisfacer los objetivos del sistema, software de seguridad.
Prueba de la documentación	Software de seguridad relacionado con la prueba de planificación, diseño de pruebas, casos de prueba, procedimientos de prueba, y los informes de ensayo.
Software verificación y validación	Información sobre cómo la seguridad del software será verificada y validada. El software de análisis relacionado con la seguridad para llevar a cabo debe ser especificado. El método (s) para garantizar la trazabilidad de los requisitos de seguridad a las especificaciones, la aplicación y el software relacionado con la seguridad de los casos de prueba se especificarán
Informe de verificación de seguridad y validación	Información sobre los resultados del software relacionado con la seguridad actividades de verificación y validación serán registrados y reportados.
Documentación del software de usuario	La información que puede ser importante para la instalación, uso, mantenimiento, y / o retiro del sistema

Tabla 200.Requisitos de documentación que deberán cumplirse para todo el software de seguridad crítica

Registros de seguridad de programa de software

Se identificarán los registros de seguridad de programas de software que se generen, mantengan y conserven en el proyecto. En el plan de gestión se identificarán los responsables de la generación de registros, retención y las tareas de mantenimiento. Los registros de seguridad del programa de software incluyen los siguientes:

- Resultados de los análisis, incluyendo V & V, realizado sobre los requisitos, diseño, documentación de código, prueba, y otras técnicas.
- La información sobre problemas de seguridad sospechosos o confirmados en el sistema preliminar o una versión.
- Los resultados de las auditorías realizadas en las tareas del programa de software de seguridad.
- Los resultados de las pruebas de seguridad realizadas en la totalidad o parte de todo el sistema.
- Un registro de la capacitación del personal de programas de software de seguridad.
- Los resultados de las certificaciones realizadas.

Figura 138.Registros de seguridad de programa de software

Actividades de gestión de configuración del software

Se describen las disposiciones para asegurar la gestión de configuración, que cumpla con los requisitos que sean necesarios para la seguridad de software crítico mediante lo siguiente:

- Herramientas de desarrollo de software.
- Software previamente desarrollado.
- Suministrado por un vendedor de software.
- Subcontratista software desarrollado.

Software actividades de aseguramiento de la calidad

Se describe la función de aseguramiento de la calidad del software para garantizar la correcta ejecución de las actividades clave del programa de software de seguridad. Orientación sobre la planificación de la garantía de calidad del software y la preparación de planes de aseguramiento de la calidad. El plan de gestión incluirá, como mínimo, una descripción de cómo:

- El Plan está preparado, aprobado, implementado, ha cambiado, y coherente con los documentos predecesores.
- Las recomendaciones técnicas resultantes de las tareas de seguridad de software son revisadas, consideradas por la autoridad de control de cambios, y, cuando proceda, aplicadas.
- Las revisiones y auditorías a las preocupaciones de seguridad de software, los requisitos, pautas y el proceso de certificación.
- La realización del programa de seguridad de software será monitoreado.

Figura 139.Descripción de las actividades del aseguramiento de la calidad

Software de verificación y validación

Se especificarán los resultados de cada actividad del ciclo de vida y se compararán con los requisitos del sistema de seguridad y análisis de riesgos del sistema para asegurar:

- Todos los requisitos de seguridad del sistema han sido satisfechos por las fases del ciclo de vida.
- No existen riesgos adicionales mediante los introducidos por el trabajo realizado durante la actividad del ciclo de vida.

Herramienta de apoyo y la aprobación

Se especifica el proceso a utilizar y los criterios que deben aplicarse en la selección, aprobación y control de herramientas, tales como: CASE, productos, compiladores, editores, generadores de árbol de fallos, y los entornos de prueba de hardware y software.

Con el fin de disminuir la posibilidad de introducción accidental de los peligros del software de herramientas del proyecto, mediante las siguientes áreas:

- Herramienta de aprobación para su uso en el proyecto.
- La instalación de las actualizaciones a las herramientas previamente aprobadas.
- El retiro de un instrumento previamente aprobado.
- Identificación de las limitaciones que pueden imponerse sobre el uso de herramientas.

Figura 140.Se describen como la posibilidad de introducción accidental de los peligros del software de herramientas del proyecto.

4.4.8.10 Seleccionando y combinando técnicas

4.4.8.10.1 Funcional y Estructuralmente

Pruebas funcionales

Las pruebas de funcionamiento se basan en la idea de que cualquier programa que puede ser considerado como una función en los valores de su dominio de entrada a los valores en su rango de salida.

Esta noción se utiliza cuando el sistema es considerado como caja negra. Esto lleva a las pruebas de caja negra, al contenido de la aplicación que no se conoce, y la función de la caja negra se entiende por completo en términos de sus entradas y salidas. El cuadro negro es muy eficaz para el conocimiento fundamental para la orientación de objetos.

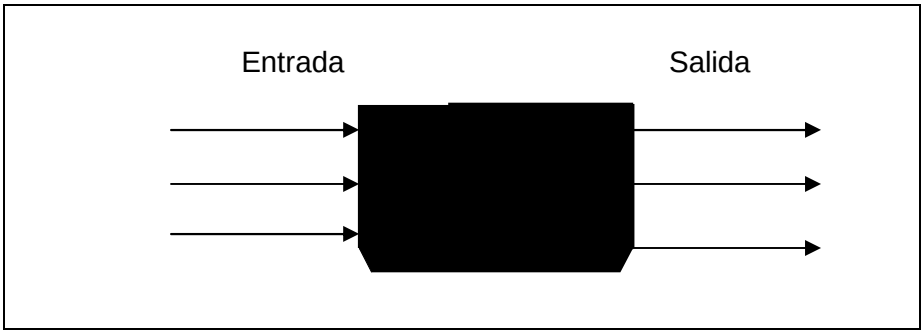


Figura 141.Cuadro negro

Con el enfoque funcional para la identificación de casos de prueba, la única información que se utiliza es la especificación del software. Hay dos ventajas claras en los casos de pruebas funcionales:

- Son independientes: cómo el software se lleva a cabo, los cambios de implementación, los casos de prueba siguen siendo útiles, desarrollo de casos de prueba mediante la ejecución que se puede realizar en

paralelo. En el lado negativo, los casos de pruebas funcionales con frecuencia adolecen dos problemas:

- No puede haber problemas entre los casos de prueba, dada la posibilidad del software no probado.
- Y esto se ve agravado por la posibilidad de brechas de software no probado.

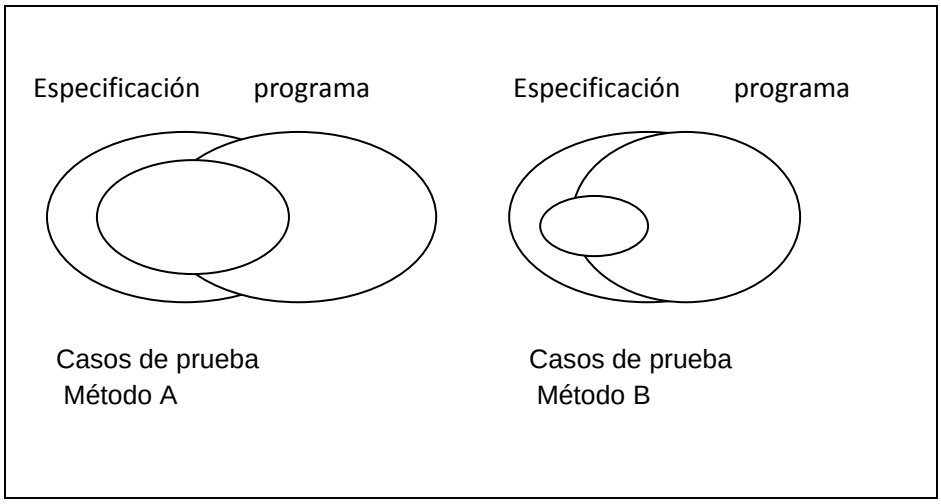


Figura 142.Comparación de métodos de prueba funcional

En la figura 142, muestra los resultados de casos de prueba identificados por dos métodos funcionales: Un método A identifica un mayor número de casos de prueba que el método B. Ambos métodos, prueban el conjunto de casos de prueba de cada uno de los métodos que contienen una conducta especificada.

Dado que los métodos funcionales se basan en el comportamiento específico, es difícil imaginar que estos métodos de identificación mediante el comportamiento no se especifica.

- En la segunda parte vamos a examinar los enfoques principales de las pruebas funcionales, incluyendo análisis de valores en la frontera, pruebas de robustez, análisis en el peor de los casos, pruebas de valor especial, clases de equivalencia de entrada (dominio), clases de equivalencia salida (rango), y pruebas tabla Decisión.

  El hilo conductor de estas técnicas es que todos se basan en la información de definición del elemento que se prueba. La base matemática presentada en las especificaciones formales se aplica principalmente a los enfoques funcionales.

Pruebas estructurales

Las pruebas estructurales es el otro enfoque fundamental para la prueba de identificación de caso de prueba. Para contrastar con las pruebas funcionales, a veces se llama caja blanca. Las pruebas han sido objeto de una teoría

bastante fuerte. Para entender las pruebas estructurales, nos basamos en los conceptos de la teoría de grafos.

Con estos conceptos, el probador puede describir exactamente lo que se está probando. Debido a su gran base teórica, las pruebas estructurales se prestan a la definición y el uso de métricas de cobertura de prueba. Las métricas de cobertura de pruebas proporcionan una manera de indicar explícitamente la medida en que ha sido un elemento de software probado, y esto a su vez, facilita la administración de las pruebas más significativas.

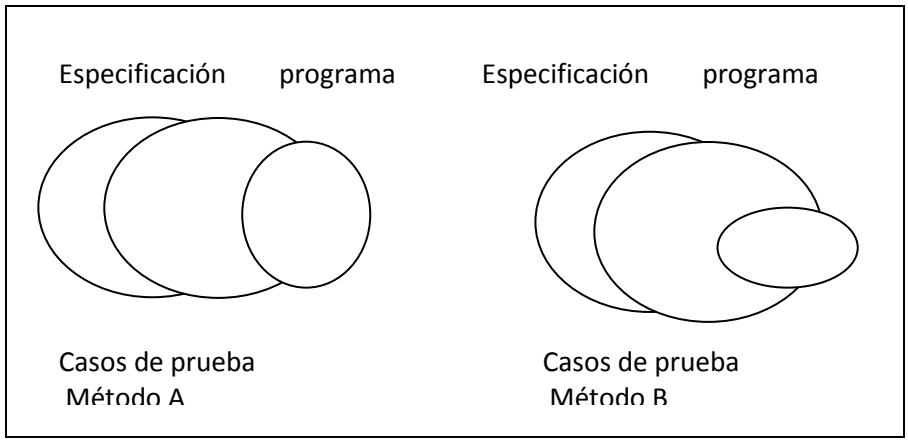


Figura 143.Comparación de los métodos de prueba estructurales

La Figura 143, muestra los resultados de casos de prueba mediante dos métodos estructurales. El método A identifica un conjunto mayor de casos de prueba que el método B.

Las pruebas estructurales proporcionan aspectos importantes para desarrollar una respuesta. Se debe tener en cuenta, para ambos métodos, que el conjunto de casos de prueba es completamente integrado en el conjunto de comportamiento programado.

4.4.9 Conclusiones

Destacaremos los siguientes puntos:

- Los modelos gráficos que combinan probabilidad y teoría de grafos, están siendo aplicados a problemas de la ingeniería del software donde la incertidumbre está presente.

- Un método de generación de casos de prueba usan una determinada técnica o conjunto de técnicas que son generados a partir de una fuente de información y con el objetivo de que los casos de prueba generados cumplan un criterio de suficiencia.

- Los procesos manuales que generan los casos de prueba suelen perder eficacia y eficiencia a medida que el sistema va creciendo. Esto es un problema y para paliarlo se utilizan técnicas que generen

automáticamente los casos de prueba según un determinado criterio de suficiencia.

- La importancia de la selección de un método con respecto a otro para la generación de casos de prueba debería desarrollarse atendiendo en primer lugar a las necesidades mediante un determinado criterio de suficiencia en el cual se pretende llevar a cabo para cumplir y cuál es la fuente de información que se dispone para generar dichos casos.

  En segundo lugar se deberá seleccionar aquel método que obtiene los resultados esperados atendiendo a cómo es el software bajo prueba y a cuáles son los resultados obtenidos en la prueba cuya características son similares; pero a veces esto no es siempre posible al no existir suficientes datos comparativos que permitan decidir que método es mejor que otro.

4.4.10 Medidas de las Pruebas

4.4.10.1 Evaluación de un programa

4.4.10.1.1 Medidas para ayudar en la planificación y diseño de pruebas de programas

En relación con las medidas y planificación de diseño de pruebas de programas se encuentran los siguientes enfoques:

- Las nuevas medidas de seguridad

  - Tiempo de falla próximo (TF): TF (t) es el tiempo previsto para los fallos Ft próximos a ocurrir, cuando el tiempo actual es t.

    TF (t) = [(log [α / (α - β (X, t + Ft.))])/] - (t - s + 1)

  - factor de riesgo del modelo de regresión
    La CF es el espacio de la memoria acumulada según la fiabilidad y la ecuación de predicción:

    CF = a × CS2 - b × c + CS

    CF= son los requisitos acumulativos de fiabilidad

    CF = d × (exp (x e IC))

RF (factor de riesgo) son los atributos de un cambio de requisitos que pueden provocar riesgo de fiabilidad, como el espacio de memoria y problemas de requisitos.

Espacio de memoria: Es la cantidad de espacio de memoria necesaria para implementar un cambio de requisitos es decir, un cambio en los requisitos de uso de memoria en la medida en que otras funciones no tienen suficiente memoria para funcionar con eficacia, y se producen fallos.

Requisitos de los problemas: Es el número de requisitos en conflicto es decir, los requisitos de cambio en conflicto con otro cambio de requisitos, tales como los requisitos para aumentar los criterios de búsqueda de un sitio web y al mismo tiempo disminuir su tiempo de búsqueda, con la complejidad del software añadido que causan los fracasos.

- Fallos restantes

  r (tt) = (α / β) (exp - β [tt - (s - 1)])

  r (tt): Son los fallos restantes.

  tt: Es el tiempo total de prueba de un comunicado o módulo.

- Tiempo total de prueba para lograr determinados fallos restantes:

  tt: Es el tiempo previsto total desde el inicio de la prueba requerido para alcanzar un determinado número de fallos resto r (tt):

  tt = [log [α / (β [r (tt)])]]/ β + (s - 1)

  tt: Es el tiempo previsto total de la prueba para un lanzamiento o un módulo.

- Fiabilidad de la red:

  RN es la fiabilidad de la red (probabilidad de supervivencia de la red).

  RN = RC x RS

  RC: Es la fiabilidad de los nc (probabilidad de sobrevivir a todos los clientes).

  RS: Es la fiabilidad de los servidores ns (probabilidad de sobrevivir a todos los servidores).

RC= 1- $\sum_{i=1}^{nc} TC\ i$/nc*T

RS= 1- $\sum_{i=1}^{ns} TSi$ / ns* T

TCi: Es el tiempo de inactividad en el cliente i (hacia abajo debido a un fallo de software y reparación) durante el tiempo T.

TSi: Es el tiempo de inactividad en el servidor i (hacia abajo debido a un fallo de software y reparación) durante el tiempo T.

- Modificación de las medidas de seguridad:

  - Densidad de fallos: Este punto se especifica con mayor detalle en el apartado 4.4.10.1.3.

  - Índice de cobertura: Este punto se especifica con mayor detalle en el apartado 4.4.10.2.1 Medidas de la cobertura/ completitud.

  - Requisitos de cumplimiento:

    Porcentaje de errores debido a los requisitos inconsistentes = N1 / (N1 + N2 + N3) x 100

    Porcentaje de errores debido a los requisitos incompletos = N2 / (N1 + N2 + N3) x 100

    N1 es el número de requisitos incompatibles en una versión o módulo.

    N2 es el número de requisitos incompletos en una versión o módulo.

    N3 es el número de requisitos malinterpretados en una versión o módulo.

  - Tasa de fallos:

    f = Δfi/Δti es el incremento de la falta de contar/incremento de prueba o el tiempo de operación.

    Δti: Es el tiempo de prueba o de funcionamiento invertido en el período i de un una versión o módulo.

    Δfi: Es el número de fallos que se producen durante Δti en una versión o módulo.

- Conservación de las medidas de seguridad

  - Requisitos de trazabilidad

TM es la medida de trazabilidad.

TM = (R1 /R2) * 100%

Si TM ≤ 100%, entonces no hay exceso de requisitos que se aplican en una versión o módulo;

En otro caso hay requisitos en exceso en la cantidad (TM - 100)%.

TM: Es la medida de trazabilidad.

R1: Es el número de requisitos a cabo en una versión o el módulo.
R2: Es el número de requisitos originales especificados en una versión o módulo.

- Tiempo medio de fallo:

  MTTF = $\sum_i^n ti$ /n

  ti: es el tiempo medio entre fallos ith, haciendo caso omiso de los tiempos de reparación, para i ≥ 1, yn es el número de veces entre fallos.

  ti: Es el tiempo medio entre fallos ith de una versión o módulo.

4.4.10.1.2 Tipos de errores, clasificación y estadísticas

- Tipos de errores

  Especificaremos dos clases de tipos:

  - Tipos de errores comunes que se cometen en la fase de prueba

- Suponer que no se encontrarán errores. - Ejecutar las pruebas solo por el programador que ha realizado el programa. - No especificar el resultado esperado. - No probar condiciones inválidas. - No definir las entradas, acciones y salidas esperadas. - No construir un conjunto de casos de prueba completo. - No llevar a cabo un enfoque basado en el riesgo, y no concentrarse en los módulos más complejos y en los que podrían hacer el mayor daño si fallan. - No recordar que también se pueden cometer errores durante las pruebas.

Tabla 201.Tipos de errores más comunes

- Tipos de errores en las pruebas de interfaz

Tipo de error	Descripción
Abusos de interfaces	Se basa en una inadecuada utilización de la interfaz. Esto es debido al paso de parámetros erróneos entre los cuales se contemplan: se pasa en orden incorrecto, se pasan un número erróneo de parámetros, los propios parámetros son de tipo erróneo.
Malentendido de interfaces	En una llamada de un componente a otro, la especificación de la interfaz del componente invocado es malentendida por el componente que invoca y provoca un error
Errores en el tiempo	Es un error mucho más inusual y casi específico de un sistema de tiempo real que utiliza memoria compartida o una interfaz para paso de mensajes. El problema surge cuando las velocidades a las que operan el producto de datos y el consumidor de datos son distintas, lo que genera un error

Tabla 202.Tipo de errores en las pruebas de interfaces

- Clasificación

El proceso de clasificación:

La organización debe definir su proceso de clasificación de la siguiente manera:

1. El objetivo (s) que debe realizarse mediante la clasificación de los defectos y fracasos.
2. La norma de referencia (por ejemplo, como se describe en una especificación, contrato o plan) para determinar qué sistema / software / conductas constituyen un fracaso.
3. Desacuerdo o conflicto en relación con las decisiones de clasificación que se han de solventar.
4. Cuando la clasificación es para empezar y terminar en el proyecto o ciclo de vida del producto.
5. El proyecto, producto, organización o valores específicos que son elegibles para la asignación de atributos de clasificación
6. Quién es la persona adecuada para asignar valores a la clasificación de los atributos y para cada defecto y fallo descubierto, respectivamente.
7. Dónde y cómo los datos de clasificación se van a mantener.

Los defectos en clasificación: La organización debe registrar los valores de todos los atributos de defecto. El estado de un defecto puede ser

insertado, detectado o eliminado, sin embargo, es común para el seguimiento del estado del informe de defectos.

Atributo	Definición
Defecto identificador	ID único del defecto.
Descripción	Descripción de lo que falta, está mal, o es innecesario.
Estado	Estado actual dentro del ciclo de vida defecto informe.
Activos	Los activos de software (de productos, componentes, módulos, etc.) que contiene el defecto.
Elemento	El producto de software de trabajo específico que contiene el defecto.
Versión detectada	Identificación de la versión de software en el que se detectó el defecto.
Versión corregida	Identificación de la versión de software en la que se corrigió el defecto.
Prioridad	Clasificación para el procesamiento asignado por la organización responsable de la evaluación, resolución y cierre del defecto en relación con otros defectos reportados.
Gravedad	El impacto mayor (fracaso) que el defecto podría (o no) causar, según lo determine (desde la perspectiva de) la organización responsable de la ingeniería de software.
Probabilidad	La probabilidad de fallo recurrente causada por este defecto.
Efecto	La clase de requisito de que se ve afectado por un fallo causado por un defecto.
Tipo	Una clasificación basada en la clase de código dentro de la cual se encuentra el defecto o el producto del trabajo en el que se encuentra el defecto.
Modo	Una clasificación en función de si el defecto se debe a una aplicación incorrecta o representación, la adición de algo que no es necesario, o una omisión.
Detección de actividad	La actividad en la que se detectó el defecto (es decir, inspección o pruebas).
La falta de referencia (s)	Identificador del fallo (s) causado por el defecto.
Cambio de referencia	Identificador de la solicitud de cambio iniciado y acciones correctivas para corregir el defecto.
Disposición	Disposición final del informe de defectos en el cierre.

Tabla 203.Atributos de defectos

La falta de clasificación: La organización debe registrar los valores de todos los atributos de fracaso.

Atributo	Definición
ID fracaso	Identificador único para el fracaso.
Estado	Estado actual, informe de fracaso dentro del ciclo de vida.
Título	Breve descripción de la falta de elaboración de informes de resumen.
Descripción	Descripción completa de la conducta anómala y las condiciones en que se produjo, incluyendo la secuencia de eventos y / o acciones de los usuarios que han precedido al fallo.
Medio Ambiente	Identificación del entorno operativo en el que se observó el fallo.
Configuración	Los detalles de configuración como producto de referencia y los identificadores de versión.
Gravedad	Según lo determinado por (desde la perspectiva de) la organización responsable de la ingeniería de software.
Análisis	Los resultados finales del análisis de las causas de conclusión de la investigación fracaso.
Disposición	La disposición final del informe de rechazo.
Abierto por	Persona que abrió (ha presentado) el informe de fallo.
Asignado a	Persona u organización asignada para investigar la causa de fallo.
Cerrado por	Persona que cerró el informe de rechazo.
Fecha en que se observó	Fecha/hora en que el fallo fue observado.
Fecha de apertura	Fecha/hora del informe de errores que se abre (presentado).
Fecha de cierre	La fecha /hora en que se cierra el informe de errores y la disposición final se le asigna.
Prueba de referencia	Identificación de la prueba específica que se llevó a cabo (si existe), cuando se produjo el error.
Referencia incidente	Identificación de los incidentes asociados si el informe de fallo.
Referencia defecto	Identificación que se detectó a causa del fallo.
La falta de referencia	Identificación de un informe de errores relacionados.

Tabla 204.Atributos de la falta de clasificación

- Estadística

  Equivale a examinar el software de la forma en que los usuarios software lo analizan a fin de determinar un conjunto de estímulos, es decir, entradas y eventos que hacen que el software cambie de comportamiento. A cada estímulo se le asigna una probabilidad de uso con base en entrevistas con usuarios potenciales

4.4.10.1.3 Densidad de fallos

Se presentan los siguientes aspectos:

Nombre	Descripción
Definición	FD: Densidad de fallo FD= F/ KSLOC
Variables	FD = Es la densidad de fallo. F = Es el número de fallos que se encuentran y dan lugar a fallos en un nivel de gravedad especificado en los grados de libertad o módulo. KSLOC = Es el número de líneas de código ejecutables y no ejecutables de declaraciones de datos de miles de personas, por los grados de libertad o módulo.
Parámetros	Especifica el nivel de gravedad.
Aplicación	Esta medida se utiliza para realizar las siguientes funciones: Establecer la densidad deseada por el grado de severidad en los fallos Comparar la densidad calculada de fallo con el valor objetivo del grado de severidad. Utilizar la función de densidad de fallo para determinar si las pruebas se han completado.
Requisitos de los datos	F: Es el recuento del número de fallos que se encuentran y dan lugar a fallos de un nivel de gravedad especificado por los grados de libertad o módulo. KSLOC: Es el recuento del número de líneas de código ejecutable y no ejecutable en las declaraciones de datos de miles de personas, por los grados de libertad o módulo.
Las unidades de medida	F, KSLOC son números dimensionales.
Herramientas	Software de hoja de cálculo, compilador de la producción de líneas de código ejecutable y no ejecutable declaraciones de datos.

Tabla 205.Elementos de la densidad de fallos

4.4.10.1.4 Modelos de crecimiento de la Confiabilidad

La confiabilidad de un producto de programación puede definirse como la probabilidad de que un programa desempeñe una función requerida bajo ciertas condiciones específicas y durante cierto tiempo.

La confiabilidad puede expresarse en términos de exactitud, firmeza, cobertura y consistencia de código fuente. Las características de la confiabilidad pueden instrumentarse en un producto de programación, pero existe un costo asociado con el aumento del nivel de análisis, diseño, instrumentación y esfuerzo de verificación y validación que debe aportarse para asegurar alta confiabilidad.

El nivel de confiabilidad deseado debe establecerse durante la fase de planeación al considerar el costo de los fallos del programa; en algunos casos

los fallos pueden causar al usuario pequeños inconvenientes, mientras que en otros tipos de productos puede generarse gran pérdida financiera e incluso poner una vida en peligro.

Boehm describe cinco categorías en la confiabilidad y recomienda la asignación de un multiplicador para cada uno de ella:

Categoría	Consecuencia del fallo	Factor
Muy baja	Molestia menor	0.75
Baja	Las pérdidas son fáciles de recuperar	0.88
Nominal	Dificultad relativa en la recuperación	1.00
Alta	Gran pérdida financiera	1.15
Muy alta	Riesgo de una vida	1.40

Tabla 206.Factores multiplicadores de esfuerzo para ajustes por confiabilidad

4.4.10.2 Evaluación de las pruebas realizadas

4.4.10.2.1 Medidas de la cobertura/completitud

Los analizadores de cobertura son una clase de herramientas de prueba que ofrecen soporte automatizado para este enfoque de la gestión de las pruebas. El analizador utiliza la información producida por la instrumentación del código para generar un informe de cobertura.

Nombre	Descripción
Definición	TCI: Es el índice de cobertura de la prueba TCI = NR/TR
Variables	NR: Es el número de requisitos que han pasado todas las pruebas mediante los grados de libertad o módulo. TR: Es el número total de requisitos de prueba de acuerdo con los grados de libertad o módulo.
Parámetros	Ninguno
Solicitud	Determinar si los requisitos se han cubierto en las pruebas
Requisitos de datos	NR: Es el número de requisitos que han pasado todas las pruebas de acuerdo con los grados de libertad o módulo. TR es el número total de requisitos de prueba de acuerdo con los grados de libertad o módulo.
Las unidades de medida	NR, TR son los números dimensionales.

Tabla 207.Índice de grado de cobertura

4.4.11 Conclusiones

Destacaremos los siguientes puntos:

- Se planifican y diseñan los casos de prueba en el orden en que se llevan a cabo. Aunque no hay regla especifica de cómo diseñarlos.

- Diseño de las pruebas de integración de componentes: Se utiliza para verificar que los componentes interaccionan entre sí de un modo adecuado después de haber sido integrados en el sistema. Se toman como casos de prueba los casos de uso de diseño. Se utilizan el diagrama de secuencia y se diseñan combinaciones de entrada y salida del sistema que conllevan a distintas utilizaciones de las clases.

- Diseño de las pruebas del sistema: Se utiliza para que el sistema funcione correctamente de forma global. Cada prueba del sistema prueba combinaciones de casos de uso bajo condiciones diferentes. Se prueba el sistema como un todo probando casos de uso unos detrás de otros, y si es posible en paralelo. El propósito de esto es ver que cada caso de uso funciona correctamente en distintas configuraciones hardware, de carga, con varios actores a la vez, distinta manera, etc.

- Un programa es seguro contra fallos cuando se ejecuta razonablemente por cualquier usuario que lo use. Para conseguir este objetivo se han de comprobar los errores en datos de entrada y en la lógica del programa.

- Un objetivo importante en la producción de sistemas es la fiabilidad. El objetivo de crear programas fiables ha de ser crítico en la mayoría de las situaciones.

- La modificabilidad se refiere a los cambios que son controlados en un sistema, por ello un sistema se dice que es modificable si los cambios en los requisitos pueden adaptarse correctamente a los cambios en el código.

- Una característica importante en la construcción de programas es el estilo de la programación. La buena calidad en la producción de programas tiene relación con la escritura del programa, legibilidad y comprensibilidad.

- Para obtener un buen estilo de la programación, es conveniente seguir reglas de estilos para la construcción de programas claros, legibles y fácilmente modificables.

- El tratamiento de errores con frecuencia necesita acciones excepcionales que constituirán un mal estilo en la ejecución normal de un programa.

- El objetivo de la eficiencia es hacer un uso óptimo de los recursos del programa. La eficiencia ha implicado recursos de tiempo y espacio. Un

sistema eficiente es aquel cuya velocidad es mayor con el menor espacio de memoria ocupada.

- La introducción de atributos irrelevantes, redundantes en la fase de proceso de construcción del modelo puede provocar un comportamiento pobre y un incremento computacional.

4.4.12 El proceso de las pruebas

4.4.12.1 Consideraciones prácticas

4.4.12.1.1 Guías para las pruebas

Para la mayoría de las empresas la guía para un plan de pruebas es una parte muy importante en la gestión del proceso de pruebas.

Los componentes que componen una guía de pruebas son los siguientes:

Nombre	Descripción
Objetivos	Los objetivos de cada fase de prueba deben ser definidos
Los criterios de terminación	Los criterios deben ser diseñados para especificar cada fase de pruebas y deberá evaluarse si son completos.
Horarios	La agenda planifica el tiempo necesario para cada fase. Se deben indicar los casos de prueba diseñados, escritos y ejecutados. Algunas metodologías de software, tales como la programación extrema exigen que se diseñen los casos de prueba y pruebas unitarias antes de codificar la aplicación y antes de comenzarlas.
Responsabilidades	Para cada fase, las personas que diseñan, escriben, ejecutan y controlan los casos de prueba, y las personas que se comprometen a reparar los errores descubiertos, deben ser identificados. Ya que en los grandes proyectos las disputas por desgracia pueden surgir en torno si los resultados de la prueba en particular representan los errores, para no entrar en discusión deben ser identificados.
Bibliotecas de casos de prueba y las normas	En un proyecto grande, los métodos sistemáticos para identificar, escribir y almacenar casos de prueba son necesarios.
Herramientas	Las herramientas de prueba necesarias deben ser identificadas, incluyendo un plan de desarrollo, adquisición, cómo se van a utilizar, y cuando se necesitan.
Tiempo en el ordenador	Este es un plan para la cantidad de tiempo de ordenador necesario en cada fase de pruebas. Esto incluye los servidores utilizados para compilar aplicaciones, si es necesario, las máquinas de escritorio necesarias para las pruebas de instalación, los servidores web para aplicaciones basadas en web, los dispositivos de red, si

	son necesarios, y así sucesivamente.
Configuración de hardware	Si las configuraciones especiales de hardware o dispositivos son necesarios, un plan es necesario que se describen los requisitos, la forma en que se cumplen, y cuando se necesitan.
Integración	Parte del plan de pruebas es una definición de cómo el programa será reconstruido. Un sistema que contiene los principales subsistemas o programas puede ser reconstruido de forma incremental, utilizando el enfoque de arriba hacia abajo o de abajo hacia arriba, por ejemplo, los bloques de construcción son los programas o subsistemas, en lugar de los módulos. Si este es el caso, un plan de integración de sistemas es necesario. El plan de integración del sistema define el orden de integración, la capacidad funcional de cada versión del sistema, y las responsabilidades para la producción y el código que simula el funcionamiento de los componentes no existentes.
Seguimiento de los procedimientos	Los medios deben ser identificados para realizar un seguimiento de diversos aspectos del progreso de las pruebas, incluyendo la ubicación de módulos propensos a errores y la estimación de los avances con respecto al calendario, los recursos y los criterios de terminación.
Depuración de procedimientos	Los mecanismos de depuración deben ser definidos mediante un informe en el cual se ha detectado el seguimiento del progreso de las correcciones, y el añadir las correcciones al sistema. Horarios, responsabilidades, herramientas, equipo y tiempo, recursos también deben ser parte del plan de depuración.
Pruebas de regresión	Se llevan a cabo después de realizar una mejora funcional o reparación para el programa. Su propósito es determinar si el cambio ha retrocedido en otros aspectos del programa. Por lo general se vuelve a ejecutar un subconjunto de casos de prueba del programa. Las pruebas de regresión son importantes porque los cambios y correcciones de errores tienden a ser mucho más propensos al error que el código del programa original. Un plan para las pruebas de regresión, quién, cómo, cuándo, también es necesario.

Tabla 208.Componentes de una guía de prueba

4.4.12.1.2 Documentación y productos de las pruebas

Documentación mínima que se debe generar durante la fase de prueba de un ciclo de vida software, y que es:

- Plan de pruebas.
- Especificación de los requisitos para el diseño de los casos de prueba.
- Los casos de prueba incluyen, además, descripción del procedimiento de prueba y la descripción de la identificación.
- Informe de incidente de prueba.
- Resumen de pruebas.

La documentación de las pruebas es necesaria para una buena organización de las mismas, así como para asegurar su reutilización. Es fundamental para optimizar tanto la eficacia como la eficiencia de las pruebas.

A continuación se explican las partes que se deben contemplar en la documentación de pruebas

1. Introducción
2. Plan de pruebas
   Elementos a probar, alcance, enfoque, recursos, programas de tiempo, personal.
3. Diseño de pruebas
   Elementos a probar, enfoque, plan detallado.
4. Casos de prueba
   Establece los datos de entrada y los eventos.
5. Procedimientos de prueba
   Pasos para preparar y ejecutar los casos de prueba.
6. Elementos de transmisión de elementos de prueba
   Elementos a probar, localización física de resultados, persona responsable para transmitir.
7. Archivo de prueba
   Registro cronológico, localización física de pruebas, nombre del probador.
8. Informe de incidentes de pruebas
   Documentación de cualquier evento que ocurre durante la prueba que requiere mayor investigación.
9. Informe de resumen de prueba
   Resume lo anterior

Figura 144.Documentación prueba del software. Adaptado de [Eric J. Braude]

Explicación de cada sección de la documentación de prueba del software:

1. Introducción

   Explica el contexto de las pruebas y su filosofía global.

2. Plan de pruebas

   Explica cómo deben organizarse personas, software y equipo para realizar el trabajo de prueba.

3. Diseño de pruebas

   Proporciona el siguiente nivel de detalle más allá del plan de pruebas. Desglosa los elementos del software involucrados, describe el orden en el que deben probarse y nombra los casos de prueba que deben aplicarse.

4. Casos de prueba

   Consiste en un conjunto de datos y el estímulo preciso que debe aplicarse para llevar a cabo el diseño de pruebas.

5. Procedimientos de prueba

   Son los pasos detallados completos para ejecutar el plan de pruebas. Incluyen todos los procedimientos de preparación, nombre de los archivos de código fuente y objeto requeridos, archivos de salida, archivos de registro, archivo de casos de prueba e informes.

6. Elementos de transmisión de elementos de prueba

   Resumen de qué pruebas se desarrollaron, quién las hizo que versiones se usaron, etc.

7. Registro de la prueba

   Esto puede ser importante al tratar de reconstruir las situaciones cuando la prueba falla.

8. Informe de incidentes de pruebas

   Detalla las ocurrencias notorias que tuvieron lugar durante la prueba.

9. Informe de resumen de pruebas

   Resume los resultados de las actividades de prueba y aporta una evaluación del software basada en dichos resultados.

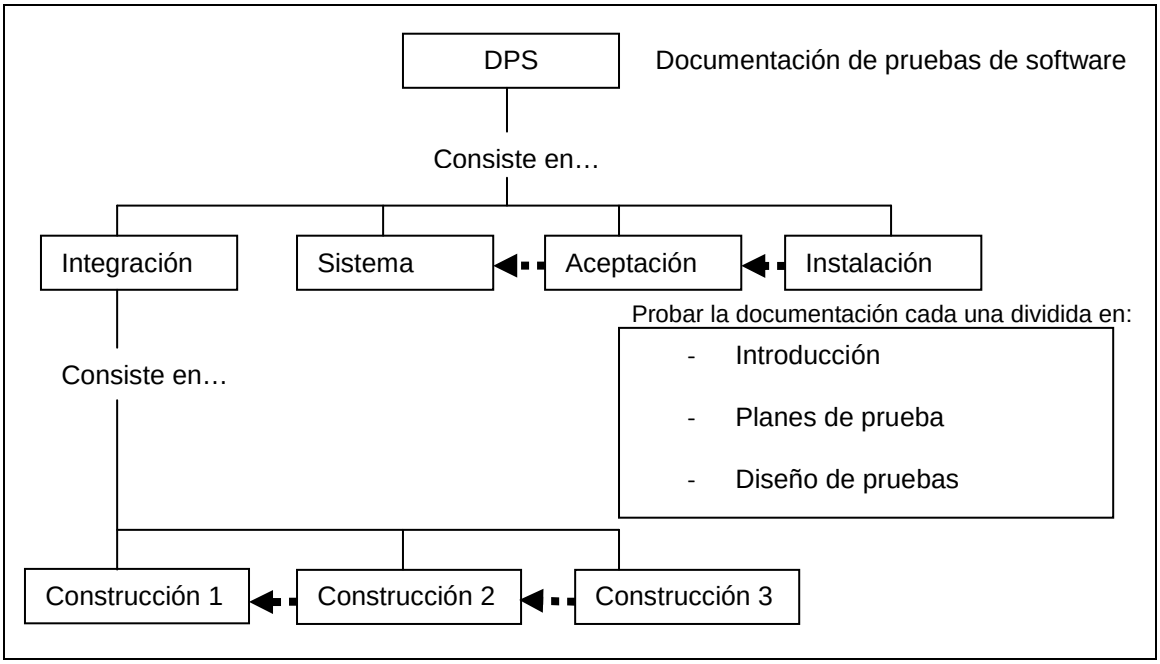


Figura 145.Organización de documentos de prueba de integración y de sistema. Adaptado de [Eric J. Brraude]

Métricas y lenguajes documentales

Uno de los aspectos menos tratados por los expertos en la documentación del software es el relativo a la necesidad de valorar la calidad de los lenguajes que

se emplean para indexar los documentos generados a lo largo del ciclo de vida del software y para desarrollar consultas sobre el repositorio que contiene todos los documentos. Es habitual referirse a los primeros como lenguajes documentales o lenguajes de indización, mientras que los segundos son lenguajes de búsqueda documental.

Los lenguajes de indización son sistemas de signos que permiten describir o identificar un documento en relación con su contenido. Utilizando este tipo de lenguajes es posible elaborar el índice de un documento como un conjunto de términos que representa el modelo conceptual de su contenido, y sobre él se realizarán después las operaciones de búsqueda, en lugar de realizarlo sobre el contenido literal del documento.

Según la definición de ISO un tesauro es el vocabulario de un lenguaje de indización controlado y organizado formalmente con objeto de hacer explicitas las relaciones, a priori entre conceptos. Las relaciones más utilizadas en un tesauro son las siguientes:

Relación	Descripción
Equivalencia	Relación biunívoca que se establece entre los descriptores y sus sinónimos
Jerárquica	Relación entre dos descriptores de los cuales uno es superior al otro por haberse establecido así en el tesauro.
Genérica	Identifica la relación entre una clase (A) y sus miembros (B,C,D…), que verifica que todos los miembros son de la clase
Parte-todo	El nombre de un descriptor inferior (B) en la jerarquía implica, en cualquier contexto, el nombre del todo al que pertenece (A), verificando que aquel es una parte de éste.
Asociativa	Relación recíproca que se establece entre términos que no son equivalentes y que no se pueden relacionar jerárquicamente

Tabla 209.Relaciones usadas en un tesauro

Existen métricas relacionadas con los lenguajes de indización del tipo tesauro que el personal dedicado a la documentación del software debe conocer si en un proyecto se decide utilizar esta modalidad del lenguaje documental. La siguiente expresión permite calcular el número de descriptores que debe contener un tesauro, y que se podría tener en cuenta en un determinado proyecto.

$$N = p.\bar{n}d.\sqrt[\bar{n}c]{\frac{\bar{n}D}{\bar{n}r.\bar{n}d}}$$

p: profundidad de la indización es decir, número de descriptores que se asignará a cada documento.

$\bar{n}$d: número medio de descriptores ligados por un operador de disyunción (OR) en las búsquedas documentales.

$\bar{n}$c : número medio de descriptores ligados por un operador de conjunción (AND) en las búsquedas documentales.

nD: número de documentos que se indizarán utilizando el tesauro.

$\bar{n}$r: número medio deseado de documentos que se recuperan por consulta.

Figura 146.Expresión, que permite calcular el número de descriptores que debe contener un tesauro

4.4.12.1.3 Equipo de Pruebas Interno vs. Equipo Independiente

El equipo de pruebas teóricamente está formado por parte del personal de aseguramiento de calidad, externo al proyecto donde se ha realizado el sistema que se está probando, por desarrolladores que han invertido en la programación del sistema, por programadores que no han intervenido en dicha codificación y por personal externo al proyecto que tenga un perfil de usuario final. En ocasiones esto no es posible bien porque no existe departamento de calidad o no se encuentran personas internas en la organización que tengan un perfil similar al usuario, etc.

En estos casos es imprescindible que el equipo de pruebas lo formen, al menos, desarrolladores involucrados en el proyecto, o desarrolladores no involucrados en el proyecto, o al menos no en la etapa de codificación, y personal externo al proyecto que no sean programadores.

Para las pruebas de unidad y de integración es suficiente con desarrolladores que hayan y que no hayan participado en la codificación del sistema. Para el resto de las pruebas, se introducirá un tercer elemento, personal ajeno al proyecto y con un perfil lo más similar posible al usuario final. En cualquier caso, si en la organización existe personal exclusivo de aseguramiento de la calidad, debe participar en todo momento.

En las pruebas de validación y las pruebas de aceptación, el usuario debe intervenir para validar los resultados del sistema. En particular las de aceptación del usuario ya que este es el que decide si está conforme con el sistema desarrollado o no.

Hay organizaciones donde existen equipos de pruebas independientes de los proyectos cuya misión es justamente, realizar las pruebas de todos los sistemas

4.4.12.1.4 Estimación Coste/Esfuerzo y otras Medidas del Proceso

Existen muchos factores que influyen en el coste de un producto de programación. El efecto de estos factores es difícil de estimar y por ende también lo es el costo del esfuerzo en el desarrollo o en el mantenimiento. Entre los factores que afectan se observa, en forma primordial, las capacidades individuales del personal asignado al proyecto y su familiaridad con el área de aplicación, la complejidad del producto, el tamaño de este, el tiempo asignado, el nivel de confiabilidad, el nivel tecnológico utilizado; disponibilidad, familiaridad y estabilidad del sistema donde se desarrolla el producto.

Factor	Descripción
Capacidad del programador	La meta era determinar la influencia relativa en la productividad de un ambiente de programación en lote (batch) y un ambiente en tiempo compartido
Complejidad del producto	Existen tres categorías para los productos de programación: Programas de aplicación, en los que se incluyen procesamiento de datos y programas científicos; programas de apoyo, como compiladores, ensambladores, sistemas de inventarios; programas de sistema como sistema de base de datos, sistemas operativos y sistemas para tiempo real.
Tamaño del producto	Un proyecto grande de programación es obviamente más caro en su desarrollo que uno pequeño. La tasa de crecimiento en cuanto al esfuerzo requerido aumenta con el número de instrucciones de código fuente que tenga un exponente un poco mayor que uno
Tiempo disponible	El esfuerzo total del proyecto se relaciona con el calendario de trabajo asignado para la terminación del proyecto, varios investigadores han estudiado la cuestión del tiempo óptimo de desarrollo y la mayoría llegan a la siguiente conclusión: los proyectos de programación requieren más esfuerzo si el tiempo de desarrollo se reduce o incrementa más de su valor óptimo.
Nivel tecnológico	En un proyecto de programación se refleja en el lenguaje utilizado, la máquina abstracta (tanto el equipo como los programadores de apoyo), las prácticas y herramientas de programación utilizadas. Se sabe que el número de líneas de código fuente escritas por día, es por completo independiente del lenguaje utilizado, y que las sentencias escritas en un lenguaje de alto nivel suelen generar varias instrucciones a nivel de máquina

Tabla 210.Factores en el coste del software

Otras medidas de seguridad: Entre otras podemos mencionar la disponibilidad

Disponibilidad

- Definición

La disponibilidad: Es la probabilidad de que el software esté disponible cuando sea necesario

Disponibilidad = MTTF / MTTF + MTTR

- Variables

MTTF y + MTTR

- Parámetros

Ninguno

- Solicitud

Calcular la probabilidad de que el software (por ejemplo, una versión o módulo) está disponible cuando sea necesario.

- Requisitos de datos

MTTF: Tiempo del fracaso medido desde el final de la reparación al fracaso siguiente.

MTTR: Tiempo medio de reparación

- Unidades de medida

MTTF y MTTR son los segundos, minutos, horas y días, según el caso.

- Herramienta

Hoja de cálculo de software

Tabla 211.Medidas de seguridad, disponibilidad

Disponibilidad de la red

- Definición

Disponibilidad de la red: Probabilidad de que la red esté disponible cuando sea necesario.

$$A = \frac{MTTFn}{MTTFn+MTTRn} = \frac{MTBFn - MTTRn}{MTBFn}$$

- Variables

  - Tiempo medio entre fallos.

MTBFn es el tiempo medio entre fallos de la red.

$$MTBFn = MTBFc * \left(\frac{nc}{nc+ns}\right) + MTBFs * \left(\frac{ns}{nc+ns}\right)$$

Promedio ponderado de MTBFc y MTBFs.

MTBFc : Es el tiempo medio entre fallos de los clientes.

$$MTBFc = \frac{T}{FC}$$

FC: Es el número total de fallos en los clientes durante el tiempo T.

nc: Es el número de nodos cliente en una red.

MTBF: El tiempo medio entre fallos de los servidores.

$$MTBFs = \frac{T}{FS}$$

FS: Es el número total de fallos en el servidor durante el tiempo T.

  - Tiempo medio de reparación

MTTRn: Es el tiempo medio de reparación de la red.

$$MTTRc = MTTRc * \left(\frac{nc}{nc+ns}\right) + MTTRc * \left(\frac{ns}{nc+ns}\right)$$

promedio ponderado de MTTRc y MTTRs

MTTRc: Es el tiempo medio de reparación para los clientes

$$MTTRc = \frac{\sum_{i=1}^{nc} mci}{FC}$$

mc_i: Es el tiempo de mantenimiento en el cliente i en el tiempo T.

MTTRs : Es el tiempo medio de reparación para los servidores.

$$MTTRs = \frac{\sum_{i=1}^{ns} msi}{FS}$$

ms_i: Es el tiempo de mantenimiento en el servidor i durante el tiempo T.

- Parámetros

nc: Es el número de nodos cliente en una red.
ns: Es el número de nodos de servidor en una red.
T: Es el tiempo previsto de funcionamiento.

- Solicitud

Disponibilidad de la evaluación de las redes.

- Requisitos de datos

fc_i: Es el número de fallos del cliente i en el tiempo T.
fs_i: Es el número de fallos de servidor i durante el tiempo T.
mcI : Es el tiempo de mantenimiento en el cliente i en el tiempo T.
msi : Es el tiempo de mantenimiento en el servidor i durante el tiempo T.
T: Es el tiempo previsto de funcionamiento.

- Unidades de medida

fci, fsi son los recuentos de errores
mc_i, ms_i, T son los segundos, minutos, horas y días, según el caso.
nc, ns son los números dimensionales

- Herramientas

Hoja de cálculo de software.

Tabla 212.Medidas de seguridad, disponibilidad de la red

4.4.12.1.5 Finalización

Comprobar la finalización, y para ello contemplamos los siguientes aspectos:

Entradas

Nombre	Descripción
La integridad y los	- Especificar una orientación general para las pruebas unitarias: Identificar áreas de riesgo que se abordarán en las pruebas, especificar limitaciones en la determinación de características, Identificar las fuentes existentes de entrada, salida, y los datos del estado. - Especificar los requisitos de integridad: Identificar las áreas a ser cubiertas por el conjunto de pruebas unitarias y el grado de cobertura necesarios para cada área, cuando se prueba una unidad durante el desarrollo de software todas las características del software deben estar cubiertas por casos de prueba, probar una unidad de práctica con un lenguaje de procedimientos, durante el desarrollo de software todas las

requisitos de terminación	instrucciones que se pueden alcanzar y ejecutar las que deben ser cubiertas por un caso de prueba o una excepción aprobada - Especificar los requisitos de terminación: Los requisitos de terminación deben incluir la satisfacción de los requisitos completos. Identificar las condiciones que podrían ocasionar la terminación anormal del proceso de prueba de la unidad. - Determinar los recursos necesarios: Estimar los recursos necesarios para el conjunto de pruebas de adquisición, ejecución inicial; considerar la posibilidad de hardware, el tiempo de acceso; identificar los recursos que necesitan preparación y las partes responsables; identificar a los responsables de las pruebas unitarias y depuración de la unidad; las necesidades de personal incluyendo el número y la duración. Especificar un horario general: especificar un horario limitado para los recursos y la disponibilidad de unidades de prueba para toda la actividad de las pruebas unitarias
Ejecución de la información	1. Ejecutar datos de entrada: - Verificar los datos de prueba - Prueba de los recursos de apoyo - configuración de los elementos de la prueba - Especificaciones y casos de prueba - Resultados de análisis de fallas (proceso de depuración) 2. Ejecutar tareas - Ejecutar pruebas: Configurar el entorno de prueba; Ejecutar el conjunto de pruebas; Registrar todos los incidentes de prueba en el apartado de notificación de problemas. - Determinar los resultados: Para cada caso de prueba; Determinar si la unidad es aprobada o no en la base de las especificaciones requeridas en el resultado de las descripciones de casos; Registro de aprobación o no en el apartado de notificación de problemas; Recoger información de seguimiento de la ejecución del resumen y adjuntar al informe. 3. Ejecutar salidas - Ejecución de la información registrada en el informe resumen de la prueba, incluidos los resultados de prueba, prueba descripciones de incidente, resultados de análisis de fallos, las actividades de corrección de fallos, corregir fallos y sus razones, y los recursos de datos, de lenguaje de procedimientos y la información de seguimiento. - Revisión de las especificaciones de prueba, si se han producido o no. - Revisión de datos de prueba, si se han producido o no.
	- Diseño de la arquitectura del conjunto de

Las especificaciones de la prueba	pruebas. - Obtener procedimientos explícitos de pruebas cuando sea necesario. - Aumentar, cuando proceda, el conjunto de especificaciones de caso de prueba basadas en la información de diseño. - Obtener las especificaciones de casos de prueba. - Completar la especificación del diseño de la prueba.
Descripción de la estructura de datos	Se describirá la estructura de datos en la cual se contemplarán los siguientes aspectos: - La descripción se cumplimentará mediante un documento en el cual se registrará cada componente de los elementos de datos. - Descripción detallada de cada uno de los componentes que forman la estructura de datos. - Pruebas de aceptación o rechazo en los datos que se adjuntará en la notificación de problema.

Tabla 213. Tomar decisiones referentes a si las pruebas son suficientes y determinar si la fase de pruebas se puede finalizar.

4.4.12.1.6 Reutilización de pruebas y patrones de pruebas

Los patrones de prueba detallan cuestiones y resuelven pruebas que pueden ayudar al tratamiento de cada una de ellas. Los patrones de prueba no proporcionan solo lineamientos eficaces de acuerdo a la iniciación de las actividades de prueba.

Los patrones de diseño proporcionan beneficios:

- Suministran un vocabulario para la solución de problemas.

- Orientan la atención en las fuerzas que hay detrás de un problema. Este enfoque es de gran importancia ya que permite a los diseñadores que trabajan en los casos de prueba un entendimiento cuando y porque se aplica a una dicha solución.

- Fomentan el pensamiento iterativo. Cada solución concibe un nuevo contexto para resolver problemas nuevos.

Estos beneficios que se han comentado anteriormente no poseen una mayor envergadura pero requieren de atención ya que nos ayudan a la solución de un problema específico. Los patrones de pruebas ayudan a comunicarse de una manera más eficaz acerca de las pruebas a un equipo del software, a discernir las fuerzas de motivación que canalizan un enfoque de manera específica para las pruebas y aproximarse al diseño de las pruebas como una actividad evolutiva en cada iteración.

Los patrones de prueba se detallan de manera similar a los patrones de diseño. A continuación se presenta la siguiente plantilla.

- Nombre del patrón.
- Resumen.

4.4.12.2 Actividades de Pruebas

4.4.12.2.1 Planificación

Contemplamos los siguientes aspectos:

Plan de la estrategia general, los recursos y la programación

Nombre	Descripción
Plan de Entradas	Proyecto de planes. Requisitos de Documentación de software.
Perfeccionar las tareas	Afinar el enfoque: Identificar la existencia de casos de prueba y procedimientos de prueba para ser considerados para su uso; identificar la existencia de los casos de prueba y procedimientos de prueba para su uso; identificar las técnicas especiales que se utilizarán para el registro de salida y validación. Especificar los requisitos de recursos especiales: identificar los recursos especiales necesarios para poner a prueba la unidad. Por ejemplo, el software que interactúa directamente con la unidad y registrar las necesidades de los recursos especiales mediante el enfoque en la especificación de la unidad de prueba de diseño. Especificar una programación detallada: especificar un calendario para la unidad de pruebas basadas en el software de apoyo, de recursos especiales, y la disponibilidad de unidades y programas de integración; registro de la programación especial mediante el enfoque en la especificación de la unidad de prueba de diseño.
Perfeccionar las salidas	Prueba específica de la unidad de planificación de la información. Unidad de prueba especial

Tabla 214.Fases del plan de la estrategia general, recursos y programación

Determinar las características de la prueba

Nombre	Descripción
Determinar las entradas	Documentación de los requisitos de la unidad de prueba. Documentación del diseño de software.
Determinar las tareas	Estudio de los requisitos funcionales: Estudio de cada requisito funcional reflejado en la documentación de la unidad de requisitos; verificar que cada requisito tiene un identificador único. Identificar los requisitos adicionales y procedimientos relacionados: identificar los requisitos funcionales que no sean por ejemplo (limitaciones de rendimiento, atributos de diseño) asociados con las características de software que pueden ser determinados mediante pruebas a nivel de unidad; identificar los procedimientos de uso o de funcionamiento asociados sólo con la unidad a ser probada. Identificar los estados de la unidad: si la documentación de la unidad específica los requisitos o implica múltiples estados (por ejemplo, inactivo, listo para recibir, procesar) de software; identificar cada estado y cada transición de estado válido. Verificar que cada estado de transición tiene un identificador único. Identificar características de entrada y salida de datos: identificar la entrada y estructuras de datos de salida de la unidad a ser probada. Para cada estructura, identificar las características, tales como las tasas de llegada, formatos, rangos de valores, y las relaciones entre los valores de los campos; Para cada característica, indicar sus rangos válidos; verificar que cada estado de transición tiene un identificador único. Seleccionar los elementos que se incluirán en las pruebas: Seleccionar las características para ser probadas; seleccionar los procedimientos asociados; estados asociados; las transiciones de estado asociado; características de los datos asociados a incluir en la prueba; seleccionar datos de entrada no válidas y válidas.
Determinar las salidas	Lista de elementos a ser incluidos en las pruebas. Unidad de requisitos de solicitud.

Tabla 215.Fases en las características de prueba

4.4.12.2.2 Generación de casos de Prueba

Contemplamos en este apartado dos aspectos:

Diseño de la serie de pruebas

Nombre	Descripción
Entradas de diseño	Documentación de los requisitos de prueba. Elementos a ser incluidos en las pruebas. Pruebas de la unidad de planificación de la información. Unidad de documentación de diseño Disponibilidad de las especificaciones de las pruebas anteriores.
Tareas de diseño	Diseño de la arquitectura del conjunto de pruebas. Con base en las características de prueba y las condiciones especificadas o implícitas en los elementos seleccionados asociadas (por ejemplo, los procedimientos, las transiciones de estado, características de los datos, etc.), el diseño de una descomposición jerárquica, conjunto de objetivos de tal manera que cada objetivo de más bajo nivel puede ser comprobado directamente por unos pocos casos de prueba. Seleccionar adecuadamente los casos de prueba existentes. Obtener procedimientos explícitos de pruebas cuando sea necesario: Combinación de la documentación de la unidad de requisitos, información sobre la planificación de pruebas, y especificación de caso de prueba, implícitamente se pueden especificar los procedimientos de prueba de unidad y por lo tanto, minimizar la necesidad de especificación explícita. Cualquiera de las opciones deben estar de acuerdo con la información requerida por la norma IEEE Std 829-2008. Obtener las especificaciones de casos de prueba: especificar los casos de prueba, las especificaciones existentes se pueden referenciar, registrar las especificaciones directamente o por referencia ya sea en una sección complementaria de las especificaciones de la unidad de prueba de diseño o en un documento separado. Cualquiera de las opciones debe estar de acuerdo con la información requerida por IEEE Std 829-2008. Aumentar, cuando proceda, el conjunto de especificaciones de caso de prueba basado en la información de diseño. Basado en información sobre el diseño de la unidad, según sea necesario actualizar la arquitectura del conjunto de prueba. Considerar las características de los algoritmos

	seleccionados y estructuras de datos internas. Completar la especificación del diseño de la prueba de la unidad de acuerdo con IEEE Std 829-2008.
Resultados de diseño	Unidad de prueba de especificación del diseño. Separar las especificaciones de procedimiento de prueba. Separar los casos de prueba. Unidad de peticiones de mejora del diseño.

Tabla 216.Fases de diseño de pruebas

Implementar el plan de refinamiento y diseño

Nombre	Descripción
Implementar las entradas	Prueba de la unidad de planificación de la información. Casos de prueba en la especificación de las pruebas de diseño y cada uno documentado por separado. Descripción de la estructura de datos. Prueba de los recursos de apoyo. Pruebas de datos de las actividades si se encuentran disponibles. Herramientas de prueba de las actividades si se encuentran disponibles.
Llevar a cabo tareas	Obtener y verificar los datos de prueba: obtener una copia de los datos de prueba existentes si existe alguna modificación; incluir datos adicionales necesarios para garantizar la coherencia e integridad de datos; verificar todos los datos; correlación entre los casos de prueba y los conjuntos de datos. Obtención de recursos especiales: recursos de apoyo. Obtener elementos de prueba: recoger todos los puntos de los cuales se compone una prueba como son: manuales, procedimientos de operación del sistema, control de datos y programas informáticos; obtener software identificado durante la planeación de pruebas que se conecta directamente con la unidad de prueba.
Implementar salidas	Verificar los datos de prueba. Prueba los recursos de apoyo. Configuración de los elementos de la prueba. . Información resumen inicial.

	Informe resumen final.

Tabla 217.Fases de implementación de pruebas

4.4.12.2.3 Desarrollo del entorno de pruebas

Orientadas a probar las interacciones entre el sistema y otros sistemas de su entorno. Para ello nos vamos a basar en herramientas de apoyo a las pruebas. En general, no hay una única clasificación de estas herramientas sino que se realiza en base a ciertos criterios como por ejemplo, su funcionalidad, es decir qué hacen y en qué fase o nivel de prueba se utilizan.

Existen herramientas con varias funcionalidades y se puede hacer uso en distintos niveles de prueba. A continuación, se detallan dos tipos de herramientas: herramientas de cobertura, herramientas de captura y reproducción.

Herramientas de cobertura	Herramientas de captura y reproducción
Monitorizan la ejecución de los casos de prueba con el objetivo de ofrecer una medida del grado de cobertura de código alcanzado por dichos casos. Se basan en instrumentar el código del software a probar añadiendo nuevas instrucciones para registrar caminos o ramas y conjuntos de sentencias ejecutadas bajo el caso de prueba. Por ende, ofrecen una medida de la cobertura de sentencias y decisiones e indican los caminos que han sido o no ejecutados bajo un determinado conjunto de casos de prueba. Si el objetivo de cobertura no se alcanza con los casos de prueba actuales, se pueden diseñar casos adicionales y ejecutar de nuevo el código bajo el control de la herramienta para determinar la nueva cobertura alcanzada	Sirven de ayuda al diseño y ejecución de los diferentes casos de prueba. La herramienta registra eventos de entrada y salida significativos durante la ejecución de la aplicación bajo prueba como por ejemplo, entradas de información, respuesta del software, etc. En modo de reproducción, dichas capturas o casos de prueba se ejecutan en escenarios reproducidos por la herramienta. Muchas de estas herramientas incorporan un lenguaje basado en scripts, de forma que se pueden especificar casos de prueba particulares y describir los pasos necesarios para su ejecución y repetición. Las herramientas de captura y reproducción son esencialmente útiles para las pruebas de regresión, ya que permiten ejecutar de nuevo los casos de prueba registrados sobre cualquier cambio en el software..

Tabla 218.Herramientas de cobertura, herramientas de captura y reproducción.

Se muestra un ejemplo de la utilización de la herramienta JUnit en la realización de pruebas unitarias de código java.

Pruebas unitarias: JUnit

JUnit es un framework que permite la automatización de pruebas unitarias sobre clases desarrolladas en el lenguaje Java. Fue creado por Erich Gamma y Kent Beck basándose inicialmente en un framework para Smalltalk llamado SUnit y que fue desarrollado por este último. JUnit es un proyecto de código abierto escrito en java y distribuido bajo la licencia Common Public License Versión 1.0.
Desde su aparición JUnit ha sido utilizado en diversos proyectos software como una herramienta capaz de asistir eficazmente al desarrollador software en la realización de pruebas unitarias. JUnit se ha convertido en la herramienta de referencia para la realización de pruebas unitarias en el mundo de desarrollo Java. Cita dedicada a JUnit:

Nunca en el campo de desarrollo software tantos le han debido tanto a tan pocas líneas de código. [MARTIN FOWLER].

MARTIN FOWLER, es actualmente uno de los autores más influyentes en el mundo de la programación orientada a objetos. Es especialista en las corrientes metodológicas ágiles y programación extrema.

Aportaciones de JUnit

Las características que la han convertido en una herramienta extremadamente útil y popular en la tarea de automatización de pruebas unitarias.

- Se encarga de solventar tareas repetitivas asociadas al proceso de pruebas tales como, organización de las clases de prueba y manejo de la situación de error. - Delimita las tareas del desarrollador, que se restringen a plasmar la información contenida en los casos de prueba en forma de código Java. - Proporciona un conjunto de métodos que facilitan la tarea de comprobación de las condiciones contenidas en los casos de prueba definidos. - Proporciona un mecanismo para ejecutar los casos de prueba de forma ordenada a la vez que mantiene información en tiempo de ejecución de los defectos software encontrado. - Consta de un número reducido de clases y métodos por lo que la curva de aprendizaje es bastante plana.

Tabla 219.Características de la herramienta JUnit

Cabe destacar dos diferencias a tener en cuenta:

- Técnicas de pruebas unitarias, es universal y se ha ido depurando con el paso del tiempo.
- La herramienta JUnit, permite poner dicha técnica en práctica de una forma eficaz para el desarrollador.

Creación de una clase de prueba

Una vez puesto el entorno de pruebas, a continuación se comienza a desarrollar el software de pruebas. Para ello conviene diferenciar entre dos conceptos que se usarán con frecuencia:

- Software de producción, no es otro que el software objetivo de las pruebas, es decir, aquel software que será entregado al cliente y al usuario final de la fase de desarrollo y cuyo funcionamiento se desea verificar.

- Software de pruebas, es el conjunto de elementos software utilizados durante la fase de pruebas, es decir, código de pruebas; herramientas y librerías usadas durante la prueba; documentación utilizada y generada; fuentes de datos, etc.

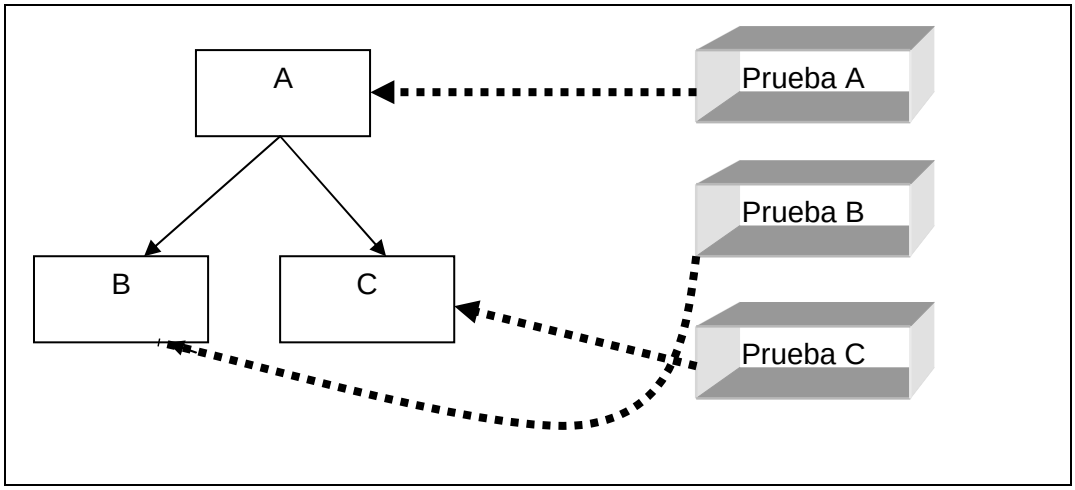


Figura 147.Relaciones entre las clases de un sistema y las correspondientes clases de pruebas unitarias. Adaptado de [Daniel Bolaños Alonso]

En la figura147, se muestran las clases correspondientes a un sistema junto con las clases creadas para realizar las pruebas unitarias de este sistema. El sistema está compuesto por las clases A, B, C, existiendo dos relaciones de asociación indicadas por flecha continua. Ya que el sistema consta de tres clases, se han definido tres clases de prueba: prueba A, prueba B, Prueba C; se van a realizar pruebas unitarias sobre cada uno de ellas.

Las flechas de línea discontinua indican la relación entre las clases de prueba y la clase a probar. Esta relación es una relación de asociación ya que instancia la clase a probar dentro de la clase de prueba. No existe relación entre las distintas clases de prueba.

Creación de una clase de prueba utilizando la herramienta JUnit

Definición	Descripción
Clase de pruebas de modo que herede de la clase	Es necesario debido a que la clase TestCase contiene una serie de mecanismos que van a ser usados por la clase de prueba como son

TestCase, perteneciente al paquete junit.framework	los métodos assert. La clase TestCase hereda de la clase Assert y por tanto todos los métodos assert(AssertNull, AsserEquals,….) van a estar disponibles en la clase TestCase
Métodos para la inicialización y liberación de los recursos utilizados durante las pruebas de la clase	Permite al desarrollador definir un método que se encargue de inicializar los recursos que se utilizarán durante la ejecución de un método de prueba, así como otro método para la liberación de dichos recursos una vez que la ejecución del método termina. Dichos métodos se llaman setUp y tearDown, se han de definir en la clase de prueba correspondiente. La gran utilidad de estos métodos es que son invocados automáticamente por JUnit antes y después de la ejecución de cada uno de los métodos de prueba
Métodos de prueba	Para cada uno de los métodos y constructores que se desea probar se ha de definir un método de prueba correspondiente. Estos métodos tendrán la funcionalidad de ejecutar uno a uno todos los casos de prueba asociados al método a probar. Secuencia de las tareas que los métodos de prueba han de realizar. - Obtener datos asociados a un caso de prueba. Varían de unos métodos a otros; por lo cual serán los parámetros de entrada con los que se invocará al método, valor de retorno, condiciones a comprobar sobre los mismos. - Invocar el método a probar con los datos del caso de prueba. Para ello es necesario crear una instancia de la clase a probar. - Comprobar ciertas condiciones sobre los valores devueltos por el método invocado así como su código de retorno y sobre el estado de otras entidades externas que pueden ser afectadas por la ejecución del método. - Ejecución del caso de prueba; ha terminado por lo que retornará al punto inicial

Tabla 220.Procedimiento de creación de una clase de prueba mediante la herramienta JUnit

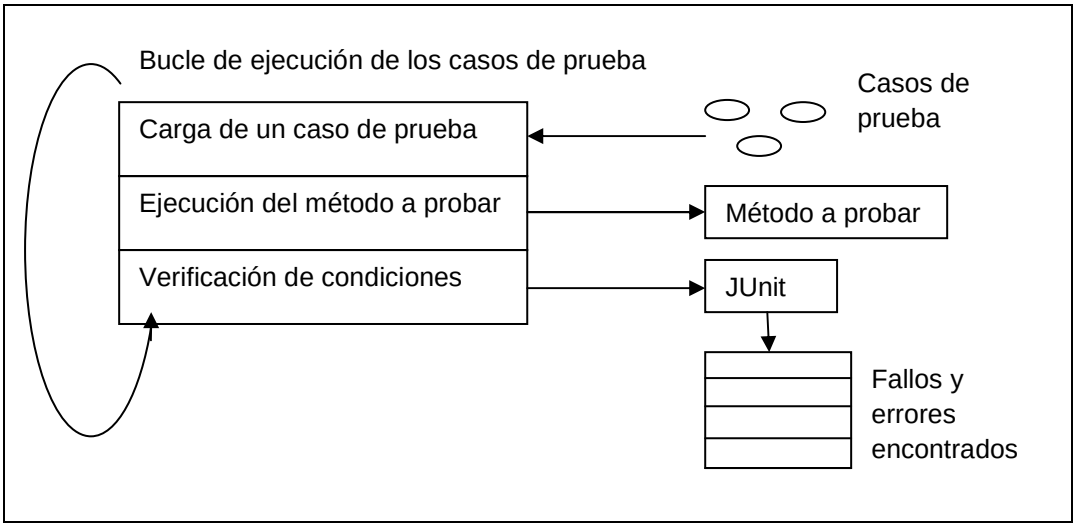


Figura 148.Ejecución de un método de prueba. Adaptado de [Daniel Bolaños Alonso]

En la figura 148, consiste en un bucle de ejecución de los casos definidos para el método a probar.

En cada iteración se cargan los datos asociados a un caso de prueba, se ejecuta mediante la herramienta JUnit mediante los métodos assert. Por lo tanto JUnit es capaz de obtener el resultado de las verificaciones y generar informe que es mostrada al final de ejecución al desarrollador.

Prueba de constructores

Los constructores en cualquier lenguaje orientado a objetos en general y en java en particular, permiten la instanciación de clases es decir, crear objetos. Los constructores son utilizados para inicializar las propiedades de un objeto que acaba de ser creado.

Los constructores presentan una serie de características respecto a los métodos que deben ser tenidas en cuenta durante la prueba:

- Carecen de valor de retorno.
- A la hora de probarlos existe un problema que es inherente a su naturaleza, y es que comprobar que se han ejecutado correctamente no es fácilmente observable.

Prueba de método get y set (obtener y asignar)

Los métodos get(obtener) y set(asignar) son típicamente utilizados en la programación orientada a objetos para acceder a las propiedades de un objeto sin romper el principio de encapsulación. Los métodos get y set permiten ocultar los detalles de implementación del objeto, en particular, permiten ocultar la forma en la que la información se almacena en el interior del objeto, es decir, constituyen un mecanismo de encapsulación de las propiedades internas del objeto.

El argumento principal que sostiene la postura en contra de los métodos get y set consiste en que se debe limitar al máximo la información intercambiada entre los objetos, porque sólo así podrá garantizar la mantenibilidad del código. Este objetivo se puede conseguir a través de un diseño modular.

El procedimiento para la prueba de los métodos get y set asociados a una propiedad es muy sencillo y se describe de la siguiente manera:

- Se instancia un objeto de la clase.
- Se almacena un valor en la propiedad con el método set de acuerdo al caso de prueba.
- Se obtiene el valor de la propiedad con el método get.
- Se comprueba que ambos valores, el obtenido y el almacenado, son idénticos para lo cual se usa un método assert, típicamente el método asserEquals.

4.4.12.2.4 Ejecución

Ejecución de entradas

Para ello establecemos los siguientes aspectos:

1. Ejecutar pruebas

   Configurar el entorno de prueba, ejecutar el conjunto de pruebas, registrar todos los incidentes de prueba en el resumen de apartado de resultados del informe de síntesis unidades de prueba.

2. Determinar los resultados

   Registro de aprobación o no de los resultados en la sección resumen de los resultados del informe de síntesis de unidades de prueba, registro de datos de los recursos en la sección resumen de actividades del informe. Al probar una unidad de práctica con un lenguaje de procedimientos, recoger información de seguimiento de la ejecución y adjuntar al informe.

Caso	Descripción
Caso 1	Un error en una especificación de prueba o datos de prueba. Corregir el defecto, registrar la corrección de errores en el informe del resumen de la prueba y volver a ejecutar las pruebas que fallaron.
Caso 2	Un fallo en la ejecución del procedimiento de prueba. Volver a ejecutar los procedimientos incorrectamente ejecutados.
Case 3	Un error en el entorno de prueba (por ejemplo, el software del sistema). Ya sea el entorno corregido, la corrección de errores se adjuntará en la notificación de problemas, y volver a ejecutar las pruebas que han fallado o prepararse para la terminación anormal y documentar el motivo para no corregir el entorno en la notificación de problemas y proceder a la verificación de la terminación.
Case 4	Un fallo en la ejecución de la unidad. Ya sea que la unidad se haya corregido, el fallo de corrección se adjuntará en la notificación de problemas y restablecer todas las pruebas o preparar para una terminación anormal y documentar el motivo para no corregir el entorno en la notificación de problemas y proceder a la verificación de la terminación.
Caso 5	Un error en el diseño de la unidad. Corregir el diseño de la unidad, modificar la especificación de prueba y los datos en su caso, registro de la corrección de errores y se adjuntará en la notificación de problemas y volver a ejecutar todas las pruebas o preparar para una terminación anormal y documentar el motivo para no corregir el entorno en la notificación de problemas y proceder a la verificación de la terminación.

Tabla 221.Seleccionar el caso aplicable y realizar las acciones pertinentes para su solución.

Ejecución de las salidas

1. Ejecución de la información registrada en el informe, resumen de la prueba incluidos los resultados de prueba, prueba de la descripción de incidencias, los resultados de análisis de errores , las actividades de corrección de errores, implementaciones de lenguaje de procedimientos, información resumida de seguimiento.

2. Especificaciones de prueba a partir de revisiones.

3. Datos de prueba producidos a partir de revisiones.

4.4.12.2.5 Evaluación de los resultados

La forma óptima de valorar la experimentación controlada en ingeniería del software consiste en el cumplimiento de las propiedades de los estudios empíricos. Los experimentos poseen una gran validez interna y se pueden controlar las amenazas a su validez con diversos procedimientos.

Con respecto a la validez externa sucede todo lo contrario, pues hay factores que limitan la generalización de los resultados. Uno de estos riesgos viene dado por la poca experiencia por parte de los probadores que se inician en la fase de pruebas.

Asimismo es importante lograr la fiabilidad de un experimento mediante su replicación. La replicación es un mecanismo que sirve para que los experimentos no se consideren estudios aislados, sino que formen parte de familias de estudios sobre algún tema de investigación en ingeniería del software. De esta manera es posible integrar los resultados y extrapolar ese conocimiento.

Los tipos de réplicas se agrupan en tres categorías:

1. Réplicas que no hacen variar ninguna de las hipótesis de la investigación, es decir, que no se modifica ninguna de las variables independientes y dependientes del experimento original.

    - Réplicas estrictas
    - Réplicas que varían la forma en que se ejecuta el experimento

2. Réplicas que varían la hipótesis de la investigación

    - Réplicas que modifican las variables independientes
    - Réplicas que modifican las variables dependientes
    - Réplicas que modifican las variables de contexto en el entorno en que se evaluó la solución.

3. Réplicas que amplían la teoría; es decir, se realizan varios cambios para comprobar si los resultados permanecen.

Tabla 222.Tipos de réplicas

4.4.12.2.6 Notificación de Problemas/Diario de pruebas

El propósito de la notificación de problemas es documentar cualquier caso que se produce durante el proceso de prueba que requiere una investigación.

A continuación se especifica un esquema de la notificación de problemas:

- Introducción.
- Documento de identificación.
- Alcance.
- Referencias.
- Detalles
- Anomalía descubierta.
- Descripción de la anomalía.
- Impacto.
- Originador de evaluación.
- Descripción de la acción correctiva.
- Estado de la anomalía.
- Conclusiones y recomendaciones.
- Procedimientos de documentar el cambio y la historia.

Diario de pruebas

El diario de pruebas pone de manifiesto todos los hechos relevantes o no ocurridos durante la ejecución de las pruebas.

Su estructura se compone de:

- Identificador.
- Descripción de la prueba: elementos probados y entorno de la prueba.
- Anotación de datos sobre cada hecho ocurrido incluido el comienzo y fin de la prueba:

    - Fecha y hora.
    - Identificador de informe de incidente.

- Otras informaciones.

4.4.12.2.7 Seguimiento de los defectos

Consideramos los aspectos de tolerancia a defectos:

Nombre	Descripción
Detección de defectos	El sistema debe ser capaz de detectar un defecto que puede conducir a un fallo de ejecución en el sistema, es decir validar el estado del sistema que se encuentra y es consistente.
Evaluación de los daños	Detecta las partes que han sido afectadas por el defecto
Recuperación de defectos	El sistema debe restablecer su estado a un estado seguro, es decir, conocido
Reparación de defectos	Muchos defectos se manifiestan como estados transitorios. La reparación de defectos implica modificar el sistema para que no vuelva a aparecer el defecto.

Tabla 223.Aspectos en la tolerancia a defectos

En los sistemas que tienen los requisitos más altos de fiabilidad y disponibilidad, se necesita emplear redundancia y diversas aproximaciones de prevención de defectos y de tolerancia a defectos.

La tolerancia a defectos es detectar que un defecto, es decir el estado del sistema erróneo, ha ocurrido u ocurrirá a menos que se realice inmediatamente una acción. Para desarrollarlo se necesitan conocimientos cuando el valor de una variable de estado es fraudulento o las relaciones entre variables de estado no se conservan.

Para ello se necesita definir restricciones de los estados que determinan las condiciones que deben cumplir los estados que son legales. Si los predicados son falsos, entonces ha tenido lugar un defecto.

Tipo	Descripción
Detección de defectos preventiva	El mecanismo de detección de defectos se inicia antes de que se produzca un cambio en el estado.
Detección de defectos retrospectiva	El mecanismo de detección de defectos se inicia después de que el estado del sistema ha sido cambiado para comprobar si ha tenido lugar un defecto.

Tabla 224.Tipos de detección de defectos

La detección de defectos preventiva se puede utilizar cuando las restricciones de los estados que han sido definidas se aplican a variables de estado individuales; evita la sobrecarga de la reparación de un defecto, ya que el estado del sistema siempre será válido. Por tanto el sistema debe mantener un mecanismo para seguir su actividad ante la presencia de un estado incorrecto si se quiere evitar un fallo de ejecución del sistema.

Un ejemplo java, la forma más fiable de implementar la detección de defectos preventiva es constatar de manera explícita la presencia de defectos y utilizar el manejo de excepciones en el lenguaje para señalar que un estado erróneo del sistema ha sido detectado.

La detección de defectos y de evaluación de daños depende de la representación de los estados del sistema y del tipo de aplicación. Las técnicas de evaluación de daños incluyen:

- Utilización de comprobaciones de código y sumas de verificación en las comunicaciones de datos y comprobaciones de dígitos en datos numéricos.

- Utilización de enlaces redundantes en estructuras de datos que contienen punteros.

- Utilización de temporizadores en sistemas concurrentes.

Los medios deben ser identificados para realizar un seguimiento de diversos aspectos del progreso de las pruebas, incluyendo la ubicación de módulos propensos a errores y la estimación de los avances con respecto al calendario, los recursos y los criterios de terminación.

4.4.12.3 Conclusiones

Destacaremos los siguientes puntos:

- En la ingeniería del software se enfatiza la necesidad de incluir experimentos controlados como mecanismos para evaluar procesos de desarrollo y productos software. Por ende, los investigadores pueden

conocer cuáles son los diseños que se ajustan a los estudios empíricos. Por lo tanto cuanto mayor es el número de factores, mayor es la complejidad del diseño, y en consecuencia mayor dificultad para analizar los resultados.

- Hemos tratado la documentación del software y en particular, métricas asociadas a ellas; estas métricas pueden ser de diferente tipo y permiten cuantificar algunos aspectos relacionados con la calidad de la documentación y de los lenguajes documentales empleados en la indización y búsqueda de los documentos que se generan a lo largo del ciclo de vida del software. Las métricas tratadas se refieren a la documentación como producto generado durante un proyecto de desarrollo de software, aunque existe la posibilidad de considerar la documentación como proceso, como uno de los procesos del ciclo de vida IEEE 12207-2008, se podrían establecer métricas asociadas al proceso que son similares a las que se usan en otros procesos del ciclo de vida y servirán para cuantificar aspectos relacionados con los recursos para llevarlo a cabo, o con el esfuerzo y tiempo requerido.

- Uno de los aspectos más importantes es el uso de herramientas de apoyo a los procesos de prueba. Estas pueden ser utilizadas en diferentes niveles desde las pruebas unitarias hasta las pruebas de regresión.

- La detección de errores, no es una tarea sencilla. Por ello es recomendable la técnica y prácticas las cuales resultan aún más efectivas si se combinan con la realización de otras actividades de aseguramiento de la calidad.

- Los procesos de prueba como cualquier otra tarea de proyecto, deben ser planificados y supervisados con el mayor detalle posible, con el objetivo de maximizar su rendimiento y por tanto, elevar la calidad final del producto.

- Para facilitar la comunicación entre todas las personas implicadas como son: probadores y jefes de proyectos, los informes de prueba pueden ser sustituidos por sistemas automáticos de seguimiento de errores o *bugtracking*. Ejemplo Bugzilla es un programa de *bugtracking* que permite gestionar simultáneamente las pruebas de distintos productos sobre los cuales pueden estar trabajando decenas de personas simultáneamente con diferentes niveles de acceso. Bugzilla permite introducir errores recién añadidos, priorizarlos y actualizarlos según se van descubriendo los defectos que los provocan y que son posteriormente corregidos.

- La documentación es una parte importante de cualquier paquete software y a su vez su desarrollo es una pieza fundamental en la ingeniería del software.

- Sintetizar los resultados obtenidos para su correcta interpretación.

- Los métodos de estimación del software han sufrido una importante evolución en los últimos años. Se ha pasado de procedimientos tan poco formales como la opinión de expertos hasta complejos modelos matemáticos de estimación que relacionan múltiples variables.

- Hoy en día, las tablas de decisión, los árboles de decisión, los algoritmos genéticos y otras técnicas de aprendizaje automático se están usando con éxito en la obtención de modelos de estimación de las características del software como tamaño, esfuerzo de desarrollo, esfuerzo de mantenimiento, defectos, facilidad de reutilización, facilidad de prueba etc.

4.4.13 Conclusiones Generales

- La etapa de pruebas requiere la prueba o verificación del programa de ordenador completado asegurando lo que hace, es decir, proporciona una solución al problema.

- La fase de pruebas es una parte esencial de un proyecto. Durante la fase de pruebas se necesita eliminar errores lógicos como se pueda. Se debe probar el programa con datos de entrada válidos que conllevan a una solución conocida. Los datos deben estar dentro de un rango, se deben incluir los valores en los extremos finales del rango.

- La prueba de un programa ocurre cuando se ejecuta un programa y se observa su comportamiento.

- Una prueba con éxito significa solamente que ningún error se descubrió en las circunstancias particulares probadas, pero no dice nada sobre otras circunstancias. El único medio de comprobar que un programa es correcto es probar todos los casos posibles.

- *Edsgar Dijkstra ha escrito que mientras las pruebas muestran efectivamente la presencia de errores, nunca pueden mostrar su ausencia.*

- Para obtener buenos datos de prueba se debe conocer que la salida debe mostrar un programa correcto para cada entrada de prueba. Las entradas de prueba deben incluir aquellas entradas que puedan originar más errores.

- Un programa bien diseñado manipula entrada con estilo, es decir, que el programa está exento de errores en tiempo de ejecución y no produce resultados indeseados; por el contrario el programa en la mayoría de los casos, visualiza un mensaje de error claro y solicita de nuevo los datos de entrada.

- Siempre suponer que un programa contiene errores hasta que las pruebas demuestren lo contrario.

- Comenzar la comprobación antes de terminar la codificación.

- Aunque no se ha mencionado nada en este documento acerca de las pruebas dinámicas, me resulta de gran importancia hacer mención de ella. Las pruebas dinámicas formulan el problema de generar casos de prueba con un problema de optimización, es decir, minimización y maximización que puede ser desarrollado mediante técnicas de optimización que pueden ser: metaheurística o clásicas.

- La alta calidad no se puede lograr sólo mediante la prueba del código fuente. Aunque un programa debe estar en su totalidad libre de errores, rara vez sucede en el caso de grandes productos software. En el supuesto de que los errores del código fuente fueran la única medida de calidad, las pruebas por sí solas, no podrían garantizar la ausencia de errores de un programa.

4.4.14 Siglas

ACC Auditoría de control de cambios
ADD Documento de Diseño Arquitectónico
ADR Revisión Diseño de Arquitectura
AFI Auditoría Física
AFU Auditoría funcional
CCB Consejo de Control de la Configuración del Software
CCC Comité de Control de Cambios
CDR Revisión de Diseño Crítico
DDD Documento de Diseño Detallado
DDR Revisión del diseño detallado
DDS Documento de Diseño Software
DMV Diseño de la máquina virtual
DPS Documento de Pruebas de Software
ECS Elemento de Configuración del software
ESA Agencia Espacial Europea
EMAS EcoManagement and AUdit Sheme
ERS Especificación de Requisitos Software
ICD Control de Documentos de Interfaz
IEC Registro del Estado de la Configuración del Software
OCI Cambio de orden en la ingeniería.
PACS Plan de aseguramiento de la calidad del software.
PAPS Plan de Administración del Proyecto de Software
PDR Revisión de diseño preliminar
PVVS Plan de Verificación y validación del Software
RSS Revisión de las especificaciones del software
SDD Descripción del diseño del software

SCI Elementos de configuración del software
SCM Gestión de la Configuración del Software
SCMP Plan de la gestión de la configuración del software.
SCMPR Plan de revisión de gestión de la configuración del software
SGMA Sistema de Gestión Medioambiental
SQA Aseguramiento de la calidad del software
SQAP Plan de Aseguramiento de la Calidad del Software
SRC Proceso de Petición de Cambios del Software
SRD Descripción de los Requisitos del Software
SRR Descripción de requisitos del software
USDP Proceso unificado de Desarrollo de Software
V & V Verificación y Validación del Software

Capítulo 5

5 Presupuesto

5.1 Introducción

Se ha realizado una guía (basada en estándares IEEE) para que empresas u organizaciones puedan elevar el nivel de calidad, disminuir costes y aumentar la productividad y ser una fuente importante de competitividad en el mercado.

“El presupuesto total de este proyecto asciende a la cantidad de 30.980EUROS

Leganés a 5 de Octubre de 2011

El ingeniero proyectista

Fdo. María del Pilar Martínez de Morentin Góngora**”**

5.1.1 Planificación

Para la planificación del proyecto este se dividió en 6 fases:

- Estudio Previo

  Se seleccionaron estándares (IEEE) tomados los más vigentes de las distintas fases del desarrollo del software.

- Aseguramiento de la calidad del software

  Se realiza un estudio, análisis, desarrollo

- Especificación de Requisitos del Software

Se desean cubrir las necesidades del cliente y se fijan las restricciones con las que debe trabajar el software a desarrollar.

- Descripción de diseño del software

  Se analizan las actividades que se deben realizar para llevarlo a cabo. A continuación se introducen algunos conceptos fundamentales que deben tenerse en cuenta para realizar cualquier diseño.

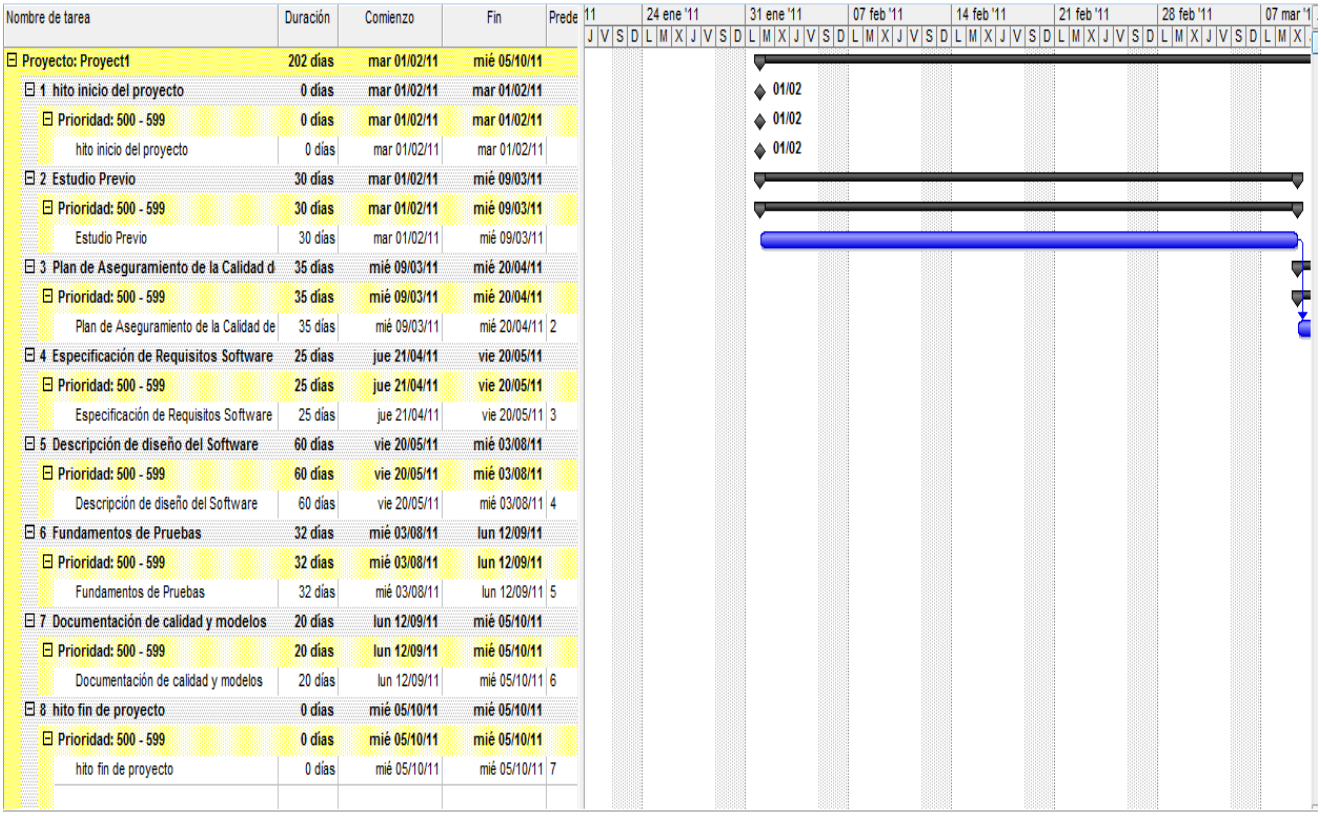
- Fundamentos de prueba del software

  Determina si los productos de una actividad del ciclo de vida determinado se ajustan a los requisitos de esa actividad, y si el producto cumple el uso previsto y las necesidades del usuario.

- Documentación de calidad y modelos

  Información recogida de documentos de calidad y modelos de evaluación de procesos.

Nota: En las distintas fases del ciclo de desarrollo del software se incluyen revisiones y mejoras



Nombre de tarea	Duración	Comienzo	Fin	Prede
Proyecto: Proyect1	202 días	mar 01/02/11	mié 05/10/11	
1 hito inicio del proyecto	0 días	mar 01/02/11	mar 01/02/11	
Prioridad: 500 - 599	0 días	mar 01/02/11	mar 01/02/11	
hito inicio del proyecto	0 días	mar 01/02/11	mar 01/02/11	
2 Estudio Previo	30 días	mar 01/02/11	mié 09/03/11	
Prioridad: 500 - 599	30 días	mar 01/02/11	mié 09/03/11	
Estudio Previo	30 días	mar 01/02/11	mié 09/03/11	
3 Plan de Aseguramiento de la Calidad d	35 días	mié 09/03/11	mié 20/04/11	
Prioridad: 500 - 599	35 días	mié 09/03/11	mié 20/04/11	
Plan de Aseguramiento de la Calidad de	35 días	mié 09/03/11	mié 20/04/11	2
4 Especificación de Requisitos Software	25 días	jue 21/04/11	vie 20/05/11	
Prioridad: 500 - 599	25 días	jue 21/04/11	vie 20/05/11	
Especificación de Requisitos Software	25 días	jue 21/04/11	vie 20/05/11	3
5 Descripción de diseño del Software	60 días	vie 20/05/11	mié 03/08/11	
Prioridad: 500 - 599	60 días	vie 20/05/11	mié 03/08/11	
Descripción de diseño del Software	60 días	vie 20/05/11	mié 03/08/11	4
6 Fundamentos de Pruebas	32 días	mié 03/08/11	lun 12/09/11	
Prioridad: 500 - 599	32 días	mié 03/08/11	lun 12/09/11	
Fundamentos de Pruebas	32 días	mié 03/08/11	lun 12/09/11	5
7 Documentación de calidad y modelos	20 días	lun 12/09/11	mié 05/10/11	
Prioridad: 500 - 599	20 días	lun 12/09/11	mié 05/10/11	
Documentación de calidad y modelos	20 días	lun 12/09/11	mié 05/10/11	6
8 hito fin de proyecto	0 días	mié 05/10/11	mié 05/10/11	
Prioridad: 500 - 599	0 días	mié 05/10/11	mié 05/10/11	
hito fin de proyecto	0 días	mié 05/10/11	mié 05/10/11	7

Figura 149. Planificación del proyecto

En la figura 149, se tiene un horario de jornada completa incluyendo las siguientes fechas:

- Estudio Previo: Del 1 de febrero al 9 de marzo, 20 días laborables.

- Aseguramiento de la calidad del software: Del 9 de Marzo al 20 de abril, 40 días laborables.

- Especificación de Requisitos del Software: Del 21 de febrero al 20 de mayo, 35 días laborables.

- Descripción del Diseño del software: Del 20 de mayo al 3 de agosto, 45 días laborables.

- Fundamentos de prueba del software: Del 3 de agosto al 12 de septiembre, 25 días laborables.

- Documentación de calidad y modelos: Del 12 de septiembre al 5 de octubre, 20 días laborables.

Los días trabajados a jornada completa serían 180 días laborables, los cuales son 20 jornadas que serían 9 meses/persona

5.1.2 Presupuesto

El proyecto ha requerido de unos materiales mínimos para su realización ya que al ser desarrollado por una persona a tiempo completo cuenta con un presupuesto moderado.

1. **Autor:** María del Pilar Martínez de Morentin Góngora

2. **Departamento**: Informática

3. **Descripción del proyecto**:

    - Titulo: Consideraciones a modo de guía a partir de estándares IEEE para calidad y buenas prácticas en desarrollo de software.
    - Duración (meses): 9 meses
    - Tasas de costes indirectos: 20%

4. **Presupuesto total del Proyecto(valores en euros):** 30.980Euros

5. **Desglose presupuestario (costes directos)**

PERSONAL

Apellidos y nombre	N.I.F	Categoría	Dedicación	Coste hombre/mes	Coste (euro)	Firma de Conformidad
Martínez, María del Pilar		Ingeniero Técnico	9	2.694,39	24249,51	
					24249,51	

meses/persona = 9

EQUIPOS

Descripción	Coste(Euro)	% Uso dedicado del proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable
Portátil	700	100	9	60 meses	105

OTROS COSTES DIRECTOS DEL PROYECTO

Descripción	Empresa	Coste imputable
Comida		200
Trasporte		180
Estándares		500
Herramientas informáticas		600

6. Resumen de costes

Presupuestos Costes Totales	Presupuestos Costes
Personal	24249
Amortización	105
Subcontratación de tareas	0
Costes de funcionamiento	1.480
Costes Indirectos	5146
Total	30.980

Capítulo 6

6 Glosario

A

- **Abstracción de datos:** El proceso de extraer las características esenciales de los datos mediante la definición de los tipos de datos y sus características funcionales asociadas y sin tener en cuenta detalles de la representación.
- **Afirmación:** Expresión lógica que especifica el estado del programa que debe existir en un conjunto de condiciones que deben satisfacer las variables del programa en un momento determinado durante la ejecución del programa. Los tipos incluyen la afirmación de entrada, la afirmación de bucle, la afirmación de la producción.
- **Anomalía:** Todo lo observado en la documentación o el funcionamiento del software que se desvía de las expectativas basadas en los productos de software previamente verificados o documentos de referencia.
- **Archivos de software de desarrollo (FAD):** Una colección de material relacionado con el desarrollo de una unidad de software o un conjunto de unidades relacionadas. Los contenidos suelen incluir los requisitos, diseño, informes técnicos, los listados de código, planes de prueba, resultados de pruebas, informes de problemas, los horarios, y las notas de las unidades.
- **Atributo:** Una característica de un elemento, por ejemplo, el color del elemento, el tamaño o tipo.
- **Aseguramiento de la calidad:** Un conjunto de actividades diseñadas para evaluar el proceso mediante el cual los productos se elaboran o fabrican.
- **Asignación:** El proceso de distribución de las necesidades, recursos, u otras entidades, entre los componentes de un sistema o programa.
- **Auditoría:** Un examen independiente de un producto de trabajo o conjunto de productos de trabajo para evaluar el cumplimiento de las especificaciones, normas, acuerdos contractuales, u otros criterios.
- **Auditoría de configuración física:** Una auditoría llevada a cabo para verificar que un elemento de configuración, como se construyó, se ajusta a la documentación técnica que lo define.

B

- **Backup:** Un sistema, componente, archivo, procedimiento, o de la persona para sustituir o ayudar a restaurar un elemento principal en el caso de un fallo o un desastre de causa externa.
- **Base de datos:** Una colección de datos interrelacionados almacenados juntos en uno o más archivos computarizados
- **Batch:** Propio de un sistema o modo de operación en que las entradas se recogen y se procesan de una sola vez, en lugar de ser procesado a medida que llegan, una vez iniciado, avanza hasta el final sin entrada adicional o sin la interacción del usuario. Contraste con: interactivo, online, en tiempo real.

C

- **Capacidad de memoria:** El número máximo de elementos que se pueden almacenar en una memoria de ordenador dado, generalmente se mide en palabras o bytes.
- **Calidad de los atributos:** Un rasgo o característica que afecta la calidad de un artículo.
- **Calidad métrica:** Una medida cuantitativa del grado en que un elemento tiene un atributo de calidad determinado.
- **Caja negra:** Sistema o componente cuyas entradas, salidas y funciones generales son conocidas, pero cuyo contenido o aplicación es desconocido o irrelevante.
- **Carácter:** Letra, dígito u otro símbolo que se utiliza para representar la información.
- **Características de software:** Una característica distintiva de un producto de software (por ejemplo, rendimiento, portabilidad y funcionalidad).
- **Casos de prueba:** Un conjunto de entradas de prueba, las condiciones de ejecución y los resultados esperados desarrollados para un objetivo particular, como el ejercicio de una ruta de programa en particular o para verificar el cumplimiento de un requisito específico.
- **CASE:** Acrónimo de la ingeniería de software asistida por ordenador.
- **Certificación:** Una garantía escrita de que un sistema o componente cumple con sus requisitos especificados y que sea aceptable para su uso operacional.
- **Ciclo:** Un período de tiempo durante el cual un conjunto de eventos se ha completado.
- **Ciclo de desarrollo de software:** El período de tiempo que comienza con la decisión de desarrollar un producto software y termina cuando el software se entrega. Este ciclo suele incluir una fase de requisitos, fase de diseño, fase de ejecución, fase de prueba y, a veces, la instalación y la fase de salida.
- **Código:** En la ingeniería de software, las instrucciones y definiciones de los datos expresados en un lenguaje de

programación o en una salida por un ensamblador, compilador, traductor u otros.

- **Cohesión:** La forma y el grado en que las tareas realizadas por un único módulo de software se relacionan unas con otras. Los tipos incluyen una coincidencia, comunicacionales, funcionales, lógicas, de procedimiento, secuencial y temporal.
- **Comando:** Una expresión que se puede introducir en un sistema informático para iniciar una acción o afectar la ejecución de un programa de ordenador.
- **Compatibilidad:** La capacidad de dos o más sistemas o componentes para cumplir con sus funciones, mientras que permite compartir el mismo hardware o entorno de software.
- **Compilador:** Un programa de computadora que traduce los programas expresados en un lenguaje de alto nivel en sus equivalentes en lenguaje de máquina.
- **Componente:** Una de las partes que componen un sistema. Un componente puede ser de hardware o software y puede subdividirse en otros componentes.
- **Componente estándar:** es un estándar que describe las características de los datos o los componentes del programa.
- **Componentes de prueba:** Las pruebas de hardware o componentes individuales de software o grupos de componentes relacionados.
- **Concepto de fase:** El período de tiempo en el ciclo de desarrollo de software en el que las necesidades del usuario se describen y evalúan a través de documentación (por ejemplo, la declaración de las necesidades, el informe de avance de planificación, memoria de la iniciación del proyecto, estudios de viabilidad, definición del sistema, documentación, reglamentos, procedimientos o políticas relevantes para el proyecto).
- **Concurrentes:** Relacionados con la ocurrencia de dos o más actividades en el mismo intervalo de tiempo, se logra mediante el intercalado de las actividades o simultánea ejecución.
- **Configuración:** La disposición de un sistema informático o un componente tal como se define por el número, la naturaleza y la interconexión de sus partes constitutivas.
- **Consecutivos:** Relacionados con la ocurrencia de dos eventos secuenciales o elementos sin la intervención de cualquier otro evento o elemento, es decir, una inmediatamente después del otro.
- **Consistencia:** El grado de uniformidad, estandarización, y la ausencia de contradicción entre los documentos o partes de un sistema o componente.
- **Constante:** Una cantidad o elemento de datos cuyo valor no puede cambiar.
- **Control de datos:** Los datos que seleccionan el modo operativo, se dirige el flujo secuencial de un programa, o de otra manera influyen directamente en el funcionamiento del software, por ejemplo, un bucle variable de control.

- **Control de diagrama de flujo:** Un diagrama que representa el conjunto de todas las posibles secuencias en que las operaciones se pueden realizar durante la ejecución de un sistema o programa. Los tipos incluyen diagrama de la caja, diagrama de flujo, diagrama de estado.
- **Control de flujo:** La secuencia en que las operaciones se realizan durante la ejecución de un programa de computadora.

D

- **Datos:** Una representación de hechos, conceptos o instrucciones de una manera adecuada para la comunicación, interpretación o procesamiento por los seres humanos o por medios automáticos.
- **Características de datos:** Un inherente, posiblemente accidental, rasgo, cualidad o propiedad de los datos (por ejemplo, las tasas de llegada, formatos, rangos de valores, o las relaciones entre los valores de campo).
- **Desarrollo de prueba:** Pruebas formales o informales realizadas durante el desarrollo de un sistema o componente, por lo general en el entorno de desarrollo por parte del desarrollador.
- **Descripción estándar:** Es un estándar que describe las características de la información sobre los productos o procedimientos previstos para ayudar a entender, probar, instalar, operar o mantener el producto.
- **Desarrollo incremental:** Una técnica de desarrollo de software en que los requisitos, definición, diseño, implementación y las pruebas se realicen en una superposición, en forma iterativa (en lugar de secuencial), lo que resulta en la terminación gradual del producto de software en general.
- **Descomposición modular:** El proceso de ruptura de un sistema en componentes para facilitar el diseño y desarrollo, un elemento de la programación modular.
- **Descripción de diseño**: Un documento que describe el diseño de un sistema o componente. Los contenidos típicos incluyen la arquitectura del sistema o componente, la lógica de control, estructuras de datos de entrada / salida de formatos, descripciones de la interfaz y algoritmos.
- **Descripción del diseño de software:** Una representación de un software creado para facilitar el análisis, planificación, ejecución y toma de decisiones. La descripción del diseño de software se utiliza como un medio para comunicar información del diseño de software, y puede ser pensado como un proyecto o modelo del sistema.
- **Diseño:** El proceso de definición de la arquitectura, componentes, interfaces y otras características de un sistema o componente.
- **Diseño estructurado:** Cualquier enfoque disciplinado para el diseño de software que se adhiere a determinadas reglas sobre la base de principios tales como la modularidad, el diseño de arriba hacia abajo, y el refinamiento de datos, las estructuras del sistema y los pasos de procesamiento.

- **Diseño orientado a objetos:** Una técnica de desarrollo de software en el que se expresa un sistema o componente en términos de objetos y las conexiones entre los objetos
- **Diseño preliminar:** El proceso de análisis de alternativas de diseño y definición de la arquitectura, componentes, interfaces, y el momento y el tamaño de las estimaciones de un sistema o componente
- **Diseño arquitectónico:** El proceso de definición de un conjunto hardware y componentes software y sus interfaces para establecer el marco para el desarrollo de un sistema informático.
- **Disponibilidad:** El grado en que un sistema o componente está operativo y accesible cuando sea necesario para su uso.
- **Documento:** Un medio, y la información grabada en él, que generalmente tiene la permanencia y puede ser leído por una persona o una máquina. Ejemplos de la ingeniería de software incluyen los planes de proyectos, especificaciones, planes de prueba, manuales de usuario.
- **Documentación:** Cualquier información escrita o gráfica, descripción, definición, especificación, generación de informes o las actividades de certificación, los requisitos, procedimientos o resultados.

E

- **Eficiencia:** El grado en que un sistema o componente cumple sus funciones designadas con el mínimo consumo de recursos.
- **Embalaje:** En el desarrollo de software, la asignación de los módulos a los segmentos que se manejan como unidades físicas distintas para su ejecución por un ordenador.
- **Ensamblar:** Para traducir un programa de ordenador se expresa en un lenguaje ensamblador equivalente al lenguaje máquina.
- **Encapsulación:** Una técnica de desarrollo de software que consiste en aislar una función del sistema o un conjunto de datos y las operaciones en esos datos dentro de un módulo y con especificaciones precisas para el módulo.
- **Entidad:** En la programación de computadoras, cualquier elemento que puede ser nombrado o denotado en un programa. Por ejemplo, un elemento de datos, la declaración de programa o subprograma.
- **Entidad-atributo:** Una característica llamada o la propiedad de una entidad de diseño. Proporciona una declaración de hecho acerca de la entidad.
- **Entidad-relación (E-R):** Un diagrama que representa un conjunto de entidades del mundo real y las relaciones lógicas entre ellas
- **Entrada:** Perteneciente a los datos recibidos de una fuente externa.
- **Especificación formal:** Una especificación escrita en una notación formal, a menudo para su uso en la prueba de la corrección.

- **Equipo de componentes de software:** Una parte funcional o lógica distinta de un elemento de configuración del equipo de software, por lo general un conjunto de dos o más unidades de software.
- **Error:** Un paso incorrecto, proceso, o de definición de datos.
- **Especificación del producto:** Un documento que especifica el diseño que las copias de la producción de un sistema o componente debe implementar.
- **Especificación de rendimiento:** Un documento que especifica las características de rendimiento que un sistema o componente debe poseer.
- **Especificación del lenguaje:** Un lenguaje, a menudo una combinación de máquinas procesables del lenguaje natural y formal, utilizada para expresar los requerimientos, diseño, comportamiento, u otras características de un sistema o componente.
- **Estándares:** Los requisitos obligatorios a emplear y aplicar como medidas para prescribir un enfoque uniforme disciplinado para el desarrollo de software, es decir, las convenciones y prácticas obligatorias en las normas de hecho.
- **Estructura de datos:** Una relación física o lógica entre los elementos de datos, diseñada para apoyar las funciones específicas de manipulación de datos.
- **Excepción:** Un evento que resulta en la suspensión de la ejecución normal del programa.

F

- **Fase de prueba:** El período de tiempo en el ciclo de vida del software en el que los componentes de un producto software son evaluados e integrados, y el producto de software es evaluado para determinar si los requisitos han sido satisfechos o no.
- **Flexibilidad:** La facilidad con que puede ser un sistema o componente modificado para su uso en aplicaciones o entornos que no sean aquellos para los que fue diseñado específicamente.
- **Flujo de datos:** La secuencia en que se lleva a cabo la transferencia de datos, uso y transformación durante la ejecución de un programa de computadora.
- **Fracaso:** La incapacidad de un sistema o componente para realizar sus funciones requeridas dentro de los requisitos de rendimiento especificados.

G

- **Garantía de calidad:** Conjunto de acciones planificadas y sistemáticas para proporcionar la confianza adecuada de un producto o servicio.

- **Generalidad:** El grado en que un sistema o componente realiza una amplia gama de funciones.
- **Gestión de la calidad:** Aspecto de la función general de la gestión que determina y aplica la política de calidad.

H

- **Hoja de verificación:** Herramienta que se utiliza para el almacenamiento de información de forma estructurada y consistente.
- **host de máquina:** Una computadora utilizada para desarrollar software destinado a otro equipo.

I

- **Identificador:** El nombre, dirección, etiqueta, o índice característico de un objeto en un conjunto de programas informáticos.
- **Implementación de requisito:** Uno de los requisitos que especifica o restringe la codificación o la construcción de un sistema o componente del sistema.
- **Ingeniería:** Es la aplicación de un enfoque sistemático, disciplinado y cuantificable a las estructuras, máquinas, productos, sistemas o procesos.
- **Ingeniería de software:** Es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software
- **Integración:** El proceso de combinación de componentes de software, el hardware, o ambas cosas en un sistema general.
- **Integridad:** El grado en que un sistema o componente impide el acceso no autorizado o modificación de programas de ordenador o de datos.
- **Interactivos:** Propios de un sistema o modo de operación en la que cada entrada del usuario causa una respuesta o acción por el sistema.
- **Interfaz.** Un componente de hardware o software que conecta dos o más componentes con el propósito de pasar información de uno al otro.

L

- **Lenguaje:** Un medio de comunicación, con la sintaxis y la semántica, que consiste en un conjunto de representaciones, convenciones y normas relacionadas que se utilizan para transmitir información.
- **Lenguaje de especificación de requisitos:** Un lenguaje de especificación con construcciones especiales y, a veces, los protocolos de verificación, para desarrollar, analizar y usar hardware, manejo de documentos o requisitos de software.

- **Lenguaje estándar:** Es un estándar que describe las características de un lenguaje utilizado para describir una especificación de requisitos, diseño, o datos de prueba.
- **Lenguaje de máquina:** Un lenguaje que puede ser reconocido por la unidad de procesamiento de una computadora.
- **Lenguaje natural:** Un lenguaje cuyas reglas se basan en el uso en lugar de ser preestablecido antes de su uso del lenguaje.

M

- **Manual de usuario:** Un documento que presenta la información necesaria para emplear un sistema o componente para obtener los resultados deseados
- **Mantenimiento correctivo:** Mantenimiento realizado para corregir defectos en el hardware o software.
- **Mantenimiento:** Facilidad con que puede ser un sistema o componente de software modificado para corregir fallos, mejorar el rendimiento u otros atributos, o adaptarse a un entorno cambiante.
- **Métricas:** Una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado.
- **Modularidad:** El grado en que un sistema o programa de ordenador está formado por componentes específicos, que un cambio en un componente tiene un impacto mínimo en los demás componentes.
- **Módulo:** Una unidad de programa que es discreta e identificable con respecto a la compilación, la combinación con otras unidades, y la carga.

O

- **Objeto:** Una encapsulación de datos y servicios que manipulan esos datos. Ver también: el diseño orientado a objetos.
- **Operando:** Una variable, constante o función en la que una operación se va a realizar.

P

- **Parámetro:** Una variable que se le da un valor constante para una aplicación específica.
- **Paquete:** Un software por separado compilable es un componente que consiste en los tipos de datos relacionados con los objetos de datos, y subprogramas. Una variable a la que se le da un valor constante para una aplicación específica.
- **Path:** En la ingeniería de software, una secuencia de instrucciones que se pueden realizar en la ejecución de un programa de computadora. En el acceso a archivos, una secuencia jerárquica de nombres de directorios y subdirectorios que especifica la ubicación de almacenamiento de un archivo.

- **Plan de desarrollo de software:** Es un plan de proyecto para un proyecto de desarrollo de software.
- **Plan del proyecto:** Un documento que describe el enfoque técnico y de gestión a seguir para un proyecto. El plan general, describe el trabajo a realizar, los recursos necesarios, los métodos a utilizar, los procedimientos a seguir, los horarios que se deben cumplir, y la forma en que el proyecto se organizará. Por ejemplo, un plan de desarrollo de software.
- **Plan de pruebas:** Es un documento que describe el alcance, enfoque, los recursos, y el calendario de actividades de la prueba prevista.
  Se identifican los elementos de prueba, las características de la prueba, las tareas de prueba, qué va a hacer cada tarea, y los riesgos que requieren planes de contingencia.
- **Plan estándar:** Es un estándar que describe las características de un esquema para el cumplimiento de los objetivos definidos o para trabajar dentro de los recursos especificados.
- **Producto estándar:** Es un estándar que define lo que constituye la integridad y la aceptación de los elementos que se utilizan o producen, formal o informalmente.
- **Programa de lenguaje de diseño:** Un lenguaje de especificación con construcciones especiales y, a veces, los protocolos de verificación, para desarrollar, analizar y documentar un programa de diseño.
- **Programa estructurado:** Un programa de ordenador construido con un conjunto básico de las estructuras de control, cada uno con una entrada y una salida.
- **Prototipado:** Una técnica de hardware y desarrollo de software en el que se desarrolló una versión preliminar de parte o la totalidad del hardware o software para permitir comentarios de los usuarios, determinar la viabilidad, o investigar el tiempo o problemas de otra índole en apoyo del proceso de desarrollo.
- **Prototipo:** Un tipo preliminar, la forma, o una instancia de un sistema que sirve como modelo para las etapas posteriores o para la versión final y completa del sistema.
- **Proceso de desarrollo de software:** El proceso por el cual las necesidades del usuario se traducen en un producto de software. El proceso consiste en traducir las necesidades del usuario en requisitos de software, la transformación de los requisitos de software en el diseño, la implementación del diseño en el código, el código de prueba y, a veces, la instalación y comprobación del software para su uso operacional.
- **Prueba de especificación de casos**: Es un documento que especifica las entradas de prueba, las condiciones de ejecución, y predice los resultados de un elemento a analizar.
- **Pruebas de funcionamiento:** Pruebas realizadas para evaluar un sistema o un componente en su entorno operativo.
- **Pruebas de integración:** Las pruebas en las que los componentes de software, hardware, o ambos se combinan y se prueba para evaluar la interacción entre ellos.

- **Prueba de la documentación:** La documentación que describe los planes o los resultados de la prueba de un sistema o componente. Los tipos incluyen la especificación de caso de prueba, prueba de notificación de incidentes, registro de la prueba, plan de pruebas, el procedimiento de prueba, informe de la prueba.
- **Prueba de la unidad:** Es un conjunto de programa de ordenador de uno o más módulos, junto con los datos de control asociados, (por ejemplo, tablas), los procedimientos de uso y operación de los procedimientos.
- **Pruebas de rendimiento:** Pruebas realizadas para evaluar el cumplimiento de un sistema o componente con los requisitos de rendimiento especificados.
- **Pruebas formales:** Pruebas realizadas de acuerdo con los planes de pruebas y procedimientos que han sido revisados y aprobados por el cliente, usuario o nivel designado de gestión
- **Puntero:** Un elemento de datos que especifica la ubicación de otro elemento de datos, por ejemplo, un elemento de datos que especifica la dirección del registro de empleados próximos a ser procesados.
- **Prueba:** Es una actividad en la que se ejecuta un sistema o componente bajo ciertas condiciones, los resultados se observan, y se realiza mediante una evaluación de algún aspecto del sistema o componente.
- **Pruebas de regresión:** Repetición de pruebas selectivas de un sistema o componente para verificar que las modificaciones no han causado efectos no deseados y que el sistema o componente sigue cumpliendo con sus requisitos específicos.

R

- **Recuperación:** La restauración de un sistema, programa, base de datos o recursos del sistema u otros. Estados en los que se pueden realizar las funciones requeridas.
- **Recursiva:** Perteneciente a una estructura de procesos y datos que se definen o se generan en términos de sí mismo.
- **Redundancia:** En la tolerancia a fallos, la presencia de los componentes auxiliares en un sistema para realizar las funciones iguales o similares a otros elementos con el fin de prevenir o recuperarse de los fracasos.
- **Rendimiento:** Es el grado en que un sistema o componente cumple sus funciones designadas dentro de las restricciones dadas, tales como la velocidad, la precisión, o el uso de memoria.
- **Requisito de desempeño:** Es un requisito que impone condiciones a una necesidad funcional, como por ejemplo el requisito que especifica la velocidad, precisión, o el uso de memoria con la que una función determinada debe llevarse a cabo.

- **Requisito físico:** Uno de los requisitos que especifica una característica física que un componente del sistema o el sistema deben poseer, por ejemplo, material, forma, tamaño, peso.
- **Requisitos de análisis:** Es el proceso de estudio de las necesidades del usuario para llegar a una definición de sistema, de hardware, o requisitos de software.
- **Requisitos de la fase:** Período de tiempo en el ciclo de vida del software en el que los requisitos para un producto de software son definidos y documentados.
- **Requisitos de las especificaciones:** Un documento que especifica los requisitos para un sistema o componente. Por lo general se incluyen los requisitos funcionales, requisitos de desempeño, los requisitos de la interfaz, los requisitos de diseño y normas de desarrollo.
- **Requisitos de las especificaciones de software:** Documentación de los requisitos esenciales (funciones, rendimiento, restricciones de diseño y atributos) del software y las interfaces externas.
- **Requisito estándar:** Es un estándar que describe las características de una especificación de requisitos.
- **Requisitos de revisión de software:** Es una revisión de los requisitos especificados para uno o más elementos de configuración de software para evaluar su capacidad de respuesta y la interpretación de los requisitos del sistema y determinar si forman una base adecuada para proceder al diseño preliminar de los elementos de configuración.
- **Requisitos de revisión:** Es un proceso o reunión en la que los requisitos para un sistema, elemento de hardware, software o elemento se presentan al personal de proyectos, gestores, usuarios, clientes u otras partes interesadas para hacer comentarios o aprobación.
- **Reutilización:** El grado en que un módulo de software o producto de trabajo puede ser utilizado en el programa de más de una computadora o un sistema de software.
- **Revisión:** Un proceso o una reunión durante la cual se presenta un producto de trabajo, o un conjunto de productos de trabajo, al personal de proyectos, gestores, usuarios, clientes u otras partes interesadas para hacer comentarios o para aprobación. Los tipos incluyen la revisión de código, revisión del diseño, la revisión de la calificación formal, revisión de los requisitos, revisión de la preparación de prueba.
- **Revisión del diseño preliminar:** Un estudio para evaluar el progreso, adecuación técnica, y el riesgo de resolución del enfoque de diseño seleccionado para uno o más elementos de configuración, para determinar la compatibilidad entre el diseño con los requisitos para el elemento de configuración, para evaluar el grado de definición y evaluar el riesgo técnico asociado a los métodos de fabricación y procesos seleccionados, para establecer la existencia y la compatibilidad de las interfaces físicas y funcionales entre los elementos de configuración y otros

elementos de equipamiento, instalaciones, software y personal, y, en su caso, para evaluar los documentos preliminares operativos y de apoyo.

S

- **Salida:** Perteneciente a los datos transmitidos a un destino externo.
- **Secuencial:** Relacionados con la ocurrencia de dos o más eventos o actividades de tal manera que uno debe terminar antes de la siguiente que comienza.
- **Semántica:** Las relaciones de los símbolos o grupos de símbolos con sus significados en una lengua dada.
- **Sencillez:** El grado en que un sistema o componente tiene un diseño y ejecución que es sencillo y fácil de entender.
- **Simulador:** Un dispositivo, programa informático, o el sistema que se comporta o funciona como un sistema dado, cuando se proporciona un conjunto de entradas controladas.
- **Simulación:** Un modelo que se comporta o funciona como un sistema dado, cuando se proporciona un conjunto de entradas controladas.
- **Sistema de ciclo de vida:** El período de tiempo que comienza cuando un sistema está concebido y termina cuando el sistema ya no está disponible para su uso.
- **Sistema de control automatizado:** Herramienta de software que acepta como entrada un programa de ordenador y una representación de sus especificaciones y produce, posiblemente con ayuda humana, una prueba o refutación de la corrección del programa.
- **Sistema de pruebas:** Pruebas realizadas en un sistema completo e integrado para evaluar el cumplimiento del sistema con sus requisitos específicos.
- **Sistema de software:** Software diseñado para facilitar la operación y mantenimiento de un sistema informático y de sus programas, por ejemplo, sistemas operativos, montadores, empresas de servicios públicos.
- **Sistema operativo:** Una colección de software, firmware, y elementos de hardware que controla la ejecución de los programas de computadora y ofrece servicios tales como la asignación de recursos del ordenador, control de tareas, control de entrada / salida, y la gestión de archivos en un sistema informático.
- **Software:** Los programas de ordenador, procedimientos y documentación asociada, y los datos relativos a la operación de un sistema informático.
- **Software característico:** Un inherente, tal vez accidental, rasgo, cualidad o propiedad del software (por ejemplo, la funcionalidad, rendimiento, atributos, restricciones de diseño, el número de estados, las líneas o ramas).

- **Software de ciclo de vida:** El período de tiempo que comienza cuando un producto de software es concebido y termina cuando el software ya no está disponible para su uso: El ciclo de vida del software normalmente incluye una fase conceptual, fase de requisitos, fase de diseño, fase de ejecución, fase de prueba, instalación y fase de salida, funcionamiento y fase de mantenimiento, y, en ocasiones, la fase de retiro.
- **Software de configuración del ítem:** Un agregado de software que está designado para la gestión de configuración y tratado como una sola entidad en el proceso de gestión de la configuración
- **Software del producto:** El conjunto completo de programas de ordenador, los procedimientos y la documentación y datos asociados, posiblemente, designado para la entrega al usuario.
- **Stub:** La ejecución del esqueleto o de propósito especial de un módulo de software, utilizados para desarrollar o probar un módulo que llama o que sea dependiente de él o un equipo de instrucción del programa sustituyendo el cuerpo de un módulo de software que está o estará definido en otra parte.

T

- **Técnicas:** Los procedimientos técnicos y de gestión que ayudan en la evaluación y mejora del proceso de desarrollo de software.
- **Tiempo real:** Propio de un sistema o modo de operación en la que el cálculo se realiza durante el tiempo real que se produce en un proceso externo, con el fin de que los resultados de cálculo se puedan utilizar para controlar, supervisar, o responder de manera oportuna para el proceso externo.
- **Tipo atómico:** Un tipo de datos, cada uno de cuyos miembros se compone de un solo elemento. Sinónimo: tipo primitivo. Contraste con: tipo compuesto.
- **Tipo de carácter:** Es un tipo de datos, cuyos miembros pueden asumir los valores de caracteres especificados y puede ser operado por los operadores de carácter, como puede ser la concatenación.
- **Tipo de datos:** Una clase de datos, que se caracteriza por los miembros de la clase y las operaciones que se pueden aplicar a ellos. Por ejemplo, el tipo de caracteres, tipo de enumeración, de tipo entero, tipo lógico, el tipo real
- **Tipo real:** Un tipo de datos, cuyos miembros pueden suponer los números reales como los valores, y puede ser operado por las operaciones aritméticas con números, como sumas, restas, multiplicación, división y raíz cuadrada
- **Traza:** Un registro de la ejecución de un programa de computadora, que muestra la secuencia de instrucciones ejecutadas, los nombres y valores de las variables, o ambas cosas. Los tipos incluyen seguimiento de la ejecución, seguimiento retrospectivo, traza subrutina, rastro simbólico, la localización variable.

- **Trazabilidad:** El grado en que una relación puede establecerse entre dos o más productos del proceso de desarrollo, especialmente los productos con un predecesor, sucesor o maestro-subordinado en relación con el otro.
- **Traza-variable:** Un registro del nombre y los valores de las variables de acceso o cambios durante la ejecución de un programa de computadora.

U

- **Unidad:** Una parte lógicamente separable de un programa de computadora.
- **Unidad de pruebas:** Las pruebas de hardware individuales o unidades de software o grupos de unidades relacionadas.
- **Utilidad:** Una herramienta de software diseñada para realizar alguna función de apoyo de uso frecuente.
- **Utilización:** Evaluación de rendimiento del equipo, una proporción que representa la cantidad de tiempo que un sistema o componente está ocupado, dividido por el tiempo que está disponible.
- **Usabilidad:** La facilidad con que un usuario puede aprender a operar, preparar los insumos e interpretar las salidas de un sistema o componente.

V

- **Validación:** Proceso de evaluación de un sistema o componente, durante o al final del proceso de desarrollo para determinar si cumple los requisitos especificados
- **Verificación:** El proceso de evaluación de un sistema o componente para determinar si los productos de una determinada fase de desarrollo reúnen las condiciones fijadas en el inicio de dicha fase.
- **Verificación y validación (V & V):** El proceso de determinar si los requisitos de un sistema o componente son completos y correctos, los productos de cada fase de desarrollo cumplen con los requisitos o condiciones impuestas por la anterior fase, y el final del sistema o componente cumple con los requisitos especificados.
- **Versión:** Una versión inicial o completa como re-edición de un documento, en lugar de una revisión resultante de la emisión de las páginas que cambian a una versión anterior.
- **Viabilidad:** El grado en que los requisitos, diseño, o los planes para un sistema o componente puede ser implementado en las limitaciones existentes.

Capítulo 7

7 Bibliografía

LIBROS

Alonso Bolaños, Daniel; Alonso Sierra, Almudena y Alarcón Rodríguez. Pruebas de Software y JUnit.: 'Un análisis en profundidad y ejemplos prácticos'. 2008.

Alonso, Fernando; Martínez, Loïc y Segovia, Francisco Javier: 'Introducción a la Ingeniería del software: Modelo de desarrollo de programas'. Delta publicaciones. Madrid 2007.

Braude, Eric J.: 'Ingeniería de Software: Una Perspectiva orientada a objetos'. RA-MA Editorial, Madrid 2003.

Campderrich Falgueras, Benet.: 'Ingeniería del Software'. 1ª Edición. EDITORIAL UOC. 2003.

Cerrada Somolinos, José A.: 'Introducción a la Ingeniería del Software''. Editorial Centro de Estudios Ramón Areces, S.A. 2000.

Dolado Cosín, José Javier.: 'Medición para la Gestión en la Ingeniería del Software'. RA-MA Editorial, Canillas, Madrid 2000.

Fairley, Richard E.: 'Ingeniería del software'. McGRAW-HILL. México 1990.

Galin, Daniel.: 'Software Quality Assurance: From theory to implementation'. Pearson Education, 2003

Jorgensen, Paul C.: 'Software Testing.: 'A Craftsman´s Approach''. 1995.

Jorgensen, Paul C. 'Software Testing.: 'A Craftsman´s Approach / Paul C. Jorgensen''. 2nd ed. 2002.

Larman, Craig.: 'UML y Patrones': Una introducción al análisis y diseño orientado a objetos y al proceso unificado. Segunda Edición. Pearson Educación, S.A., Madrid, 2003

Myers, Glenford J.: 'THE ART OF SOFTWARE TESTING / Glenford J. Myers; Revised and updated by Tom Badgett and Todd Thomas, with Corey Sandler'. 2nd ed. 2004.

Piattini, Mario G; Manzano Villalón, José A; Cervera, Joaquín y Fernández Luis.: 'Análisis y Diseño de Aplicaciones Informáticas de Gestión: Una Perspectiva de Ingeniería del software". Edición RA-MA 2004.

Helm, Richard; Johnson Ralph y Vlissides.: ‘Patrones de Diseño’. Pearson Educación, S:A., Madrid 2003.

S. Pressman, Roger.: IN ‘Ingeniería del Software’: Un enfoque práctico. 7ª Edición. Mc Graw Hill. 2010.

Sommerville, Ian.: ‘Ingeniería del Software’. 7ª Edición. PEARSON EDUCACIÓN, S.A. Madrid 2005.

Tuya, Javier; Ramos Roman, Isabel y Dolado Cosín, Javier.: ‘Técnicas cuantitativas para la Gestión en la Ingeniería del Software’. Netbiblo, S.L. 2007.

BIBLIOGRAFÍA UTILIZADA

Berzins, V., “An appraisal of program specifications”, research Directions in Software Techology, ed. P. Wegner, Cambridge, Massachutes, MIT Press, 1979.

Boehm, B.W., “Some steps towards formal and automated aids to software requirements analysis and design”, IFIP 74, 192-7. Amsterdam, 1974.

Codd, E.F., “A relational model of data for large shares data banks” Comm. ACM, 13, 377-387,1970.

Crosby, Deming, Juran y Humphrey. Padres de la calidad

Dunn, R.H., “Software Defect Removal”, McGraw-Hill,1984.

Freedman, D.P. y Weinberg G. M., “Handbook of Walkthrougs, Inspections and Technical Reviews”, Little Brown & Company, 1982.

Hoare, C. A. R., “An axiomatic basis for computer programming”, Com. ACM 1969.

Humphrey, W., “The team Software process” (TSP). Technical Report CMU/SEI-2000.

M. Juran, Joseph., “Manual de control de calidad”. 1971

Liskov, B., y Zilles, S., “Programming with abstract data types”, ACM Sigplan Notices, 1974.

McCarthy, J. “Towards a mathematical science of computation”, IFIP, Amsterdam, North Holland 1962.

Myers, G.J., “The Art of Software Testing”, Nueva York, John Wiley & Sons, 1979.

T. DeMarco. “Structured Analysis and System Specification”. Prentice-Hall, 1982.

ESTÁNDARES

INTERNATIONAL STANDARD IEEE Std 12207-2008. Systems and Software Engineering Software life cycle processes. Second edition 2008-02-01, 138 p.

INTERNATIONAL STANDARD IEEE Std 1016-2009. IEEE Standard for Information Technology Systems Design Software Design Descriptions. 20 July 2009, 40 p.

INTERNATIONAL STANDARD IEEE Std 15288-2008. Systems and software engineering System life cycle processes. Second edition 2008-02-01, 84p.

INTERNATIONAL STANDARD IEEE Std 1471-2000. Systems and software engineering Recommended practice for architectural description of software-intensive systems. First edition 2007-07-15, 34p.

lEEE Std 610.12-1990 IEEE Standard Glossary of Software Engineering Terminology. 84p.

IEEE Std 1320.1-1998 IEEE Standard for Functional Modeling Language Syntax and Semantics for IDEF0. 115 p.

INTERNATIONAL STANDARD IEEE Std 1465-1998(R2004). Adoption of International Standard ISO/IEC 12119: 1994(E) Information Technology Software Packages Quality requirements and testing. Reaffirmed 8 December 2004, 25 p.

INTERNATIONAL STANDARD IEEE Std 1465-1998(R2004) [Adoption of ISO/IEC 12119: 1994(E)]. Information Technology Software packages Quality requirements and testing. Reaffirmed 8 December 2004, 25 p.

INTERNATIONAL STANDARD IEEE Std 1028-2008 (Revision of IEEE Std 1028-1997)**.** IEEE Standard for Software Reviews and Audits. Approved 16 June 2008, 52 p.

INTERNATIONAL STANDARD IEEE Std 828-2005 (Revision of IEEE Std 828-1998). IEEE Standard for Software Configuration Management Plans**.** 12 August 2005, 27 p.

Capítulo 8

8 Conclusiones

En este capítulo recopilamos las conclusiones en el desarrollo de este proyecto. Dando una gran importancia a las aportaciones que ofrece este proyecto a las empresas u organizaciones ya que observamos deficiencias u oportunidades de mejora respecto a problemas que se repiten en el trabajo diario, por ello disponemos de una guía que permita estructurar y desarrollar un proceso de mejora continua que aborde de una manera sistemática y fiable una mayor capacidad para alcanzar una solución óptima para las empresas u organizaciones siempre con miras a la calidad, ya que hoy en día la calidad es la fuente de supervivencia de las empresas.

8.1 Objetivos alcanzados

Con este proyecto hemos alcanzado los siguientes objetivos:

- Exigencias de una mejora en la calidad de los productos y servicios, requiere de una estrategia para eliminar el mayor número de obstáculos que se pueden presentar en la creación de un producto o servicio e incrementar la efectividad y competitividad en el mercado y un mayor grado de bienestar, seguridad e integridad en las empresas u organizaciones.

- Incentivar a las empresas u organizaciones para que incrementen la calidad de sus productos o servicios e impulsen el perfeccionamiento de los sistemas de calidad en sus empresas u organización.

- Se ha desarrollado un estudio sistemático en la creación de un producto o servicio con miras a la calidad, ya que la calidad es cada vez más demandada por el mercado ya que las empresas u organizaciones que produzcan calidad tienen futuro.

8.2 Conclusiones finales

Esta guía es la solución de muchas empresas u organizaciones que pretendan crear un producto o servicio creando confianza, seguridad bienestar e innovación. Mediante el ciclo de vida del software desde la fase inicial hasta la final se han tenido en cuenta hasta los más mínimos aspectos incluidas las deviaciones o la falta de una o varias características de calidad con respecto a los requisitos especificados por el cliente, así mismo se han tenido en cuenta las características requeridas de un producto o servicio mediante los niveles de calidad, el uso correcto de los mismos, la seguridad, magnitud, métodos,

metodología, técnica, prescripciones aplicables en lo relativo a la terminología, inspecciones etc.

8.3 Vías de investigación futura

A partir de este proyecto se proponen como líneas de investigación las siguientes:

- Metodologías de mejora y herramientas que permitan mejorar las formas de gestionar y obtener mejores resultados. Ya que algunas son específicas para ciertos tipos de organizaciones y otras son universales, es decir, en general su puesta en práctica potencia los siguientes aspectos.
    - Liderazgo, actúa sobre el rol a realizar modificando su compromiso y labor.
    - Grado de formalización mediante un plan para llevarlas a cabo mediante los objetivos estratégicos y políticos de la empresa u organización.
    - El personal, introducen y modifican sobre sus actitudes, aptitudes, compromiso y comportamiento.
    - Aprovechamiento de los recursos.
    - Aumento de la eficacia y eficiencia de los procesos.
    - Mejorar los resultados en los clientes.

- Desarrollar herramientas que comprendan los siguientes aspectos:

    - Sencillez, todos pueden usarlas.
    - Aplicabilidad, en todos los niveles de la organización o empresa.
    - Utilidad, gran capacidad de análisis y mejora.

- Sintetizar los resultados obtenidos para su correcta interpretación.

- Promover una relación más estable, donde se puedan conocer mejor y con antelación las necesidades de los clientes.

- No tanto bajo el supuesto de asegurar directamente a los clientes la calidad de los productos, sino más bien en la calidad del proceso empleado en la producción como medio indirecto de asegurar un buen nivel de calidad.

- El mundo del software ha creado su propia línea de trabajo en la gestión de la calidad del software tomando las ideas básicas de la anterior, es decir, se trabaja sobre los procesos de producción del software como medio de asegurar la calidad del producto software. Por ello se pretende con esto realizar un estudio y clasificación de los distintos procesos involucrados en la producción del software bajo el enfoque de una serie de niveles de madurez. Modelos pioneros que suponen actualizaciones y variantes realizadas en su entorno como iniciativas en otros ámbitos.

8.4 Opiniones Personales

La realización de este proyecto me ha aportado afianzar mis conocimientos en el desarrollo del ciclo de vida del software y sobre todo como lo es la calidad que es el fundamento en que se basa un producto, estas fases del ciclo de vida son todas de un alcance de gran importancia ya que requieren de un estudio minucioso y detallado. Muchas cosas de estas fases las desconocía por no decir un 70% pero a medida que me iba entrando en cada una de ellas fui comprendiendo su importancia, análisis y desarrollo.

El estudio y comprensión de cada una de las fases del ciclo de vida me ha supuesto invertir mucho tiempo y dedicación ya que parte de la documentación requerida se encontraba en inglés, tuve que consultar muchas fuentes que me permitieran obtener un conocimiento para el desarrollo de esta guía. Pero a su vez me permitió indagar en el mundo de la investigación ya que para mí es un factor de gran importancia tanto intelectual como personalmente.

Muchas personas hacemos lo que hacemos para obtener un resultado en función de sus consecuencias, pero muchas veces no somos conscientes de la repercusión que esto conlleva; el seguir una buena guía te aporta seguridad, bienestar y sobre todo resultados óptimos.

En conclusión, para mí este proyecto ha sido tan productivo y eficaz que he comprendido la importancia de un buen desempeño en un producto software. Y además afianzar conocimientos para futuros proyectos, como líneas de investigación.